CONTENTS

	Page
1. Introduction	3
2. Routines for BLINK	4
2.1. Main Routine	4
2.2. Service Routines	21
2.3. Input Routines	57
2.4. Input Editing Routines	72
2.5. Deck Generation Routines	90

TABLES

Page

1-1 Location of Listings for Each Functional Block 3

1. Introduction

This attachment contains listings of the FORTRAN routines and C functions that comprise version 0 of the BLINK software routine. The listings are divided by functional block as in the description of the software routine given in Attachment VI. Section 2 of this attachment provides the listing of the code consistent with the functional blocks of Attachment VI and Table 1-1 below.

Table 1-1 Location of Listings for Each Functional Block

Section	Contents
2.1	Main Function
2.2	Service Routines
2.3	Input Routines
2.4	Input Editing Routines
2.5	Deck Generation Routines

2. Routines for BLINK

2.1. Main Routine

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Load Library Header Files - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <malloc.h>
#include <errno.h>

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Global Type Definitions - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                       } surface_usage_list;

typedef struct fuel_geometry{
                              ascii_string gds_name;
                              int latdim;
                              int nwr;
```

```
                                float cthick;
                                float asin;
                                float wgap;
                                float ngap;
                                float cradius;
                                float fsrd;
                                float cfsrd;
                                float rpitch;
                                float cod;
                                float cld;
                                float pod;
                                char frcmat[6];
                                char fcmat[6];
                              } fg_list;

typedef struct all{
                     struct all *last;
                     int basis_lattice_material_index;
                     int lattice_material_index;
                     struct all *next;
                    } augmented_lattice_list;

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Function Prototypes - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*/
void abort();
void header();
void lines();
void echo(char [],char []);
int *memory_integer(int,int,int *);
float *memory_float(int,int,float *);
ascii_string *memory_ascii_string(int,int,ascii_string *);
void readin(char[],int *,char[],int *,int *,int *,float *,int *
 ,int *,int *,int *,float *,float *,float *,float *,float *,float *
 ,float *,float *,float *,char *,char *,char [],char *,char *
 ,char *,char *,char *,char *,int *,char[],char[],float *
 ,float *,float *,float *,float *);
void redmaps(int *,int *,int *,int *,int *,int *,int *,int *,int *
     ,int *,int *,int *,int *,int *,int *,int *,int *,int *, int *
     ,int *);
void redlats(int *, int *, int*, int *);
void ldlv(int *,int *,int *,int *);
void ldlvr(int *,int *,int *,float *);
void nladd(int, ascii_string *);
void ftclose(int *);
void rblade(int *,char[],int *,int *,float *,float *,float *,float *
 ,float *,float *,float *,float *,int *,float *,float *,char[],char[]
 ,char[],char[]);
void rlattice(int *,char[],int *,float *,float *,float *,float *
 ,float *,float *,float *,float *,float *, float *,float*,char[]
 ,char[],int *,int *);
fg_list *memory_fg_list(int,int,fg_list *);
void lodct(int *,int *,int *,int *,int *,int *,int *,int *,int *
```

```
 ,int *);
void editin(char[],char [],char [],int,int,float,int,int,int,int,int
 ,int,int *,int *,int *, int *,int *,int,int,ascii_string *
 ,ascii_string *,char [],int,char[],float,float,float,char[],char[]);
void coredb_edt(int,int,int,int,float,float,float,float,float,float
 ,float,float,float,int *,char [],char [],char [],char []
 ,char [],float,float);
void bladedb_edt(int,float,float,float,float,float,float,float,float
 ,int,float,float,char[],char[],char[],char[]);
void fgds_edt(int,char[],ascii_string *,fg_list *);
void edit_ct(int,int *p);
int clattice(int *, int *,int *,int *,int *);
void strngr(int *,int *,int *,ascii_string *);
a_record *load_core_mtls(char[],int);
a_record *memory_ascii_record(int,a_record *);
usage_list *memory_usage_list(int,usage_list *);
surface_usage_list *memory_surface_usage_list(int
 ,surface_usage_list *);
void vessel_generation(float,float,float,float,float,float,float,float
 ,float,FILE *,FILE *,FILE *,int *,int *,int *,a_record *,char[]
 ,char[],char[],char[],char[],char[],int,surface_usage_list *
 ,usage_list **,float,float);
void ccmgen(int *,int *,int *,int *,int *,int *,int *,int *
 ,int *,int *,int *,int *,int *,int *,int *);
void build_control_blade(int,float,float,float,float,float,float
 ,float,float,int,float,float,char[],char[],char[],char[],usage_list *
 ,surface_usage_list *,a_record *,FILE *,FILE *,FILE *,int *,int *
 ,float,int *,int *,int *,int *);
void generate_deck(FILE *,FILE *,FILE *,FILE *);
void copy_ascii_file(FILE *,FILE *);
void echo_MCNP_deck(FILE *);
void discard_scratch_file(char []);
void memsum();
int mchar(int *,char []);
ll_material *load_fuel_material(int *,char[],char[],int,int **,float **);
void generate_lattice_model(int *,int *,FILE *,FILE *,FILE *
 ,surface_usage_list *,usage_list *,int *,fg_list *,ascii_string *
 ,int,char[],float,float,int *,a_record *,int *,int *,char[]);
void build_assemblies(int *,int *,FILE *,FILE *,surface_usage_list *
 ,float *,int **,int,int *,int,int *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
void build_control_cells(int *,int *,int,int,int *,int *,int,FILE *,FILE *
 ,float,surface_usage_list *,float,int,float,float,char[],usage_list *
 ,a_record *,int *,int *,FILE *,float,float,float,int,int *,int *);
void core_lattice_generation(int *,int *,float,float,surface_usage_list *
 ,usage_list *,int *,int,int,FILE *,FILE *,int,int *,float);
void source_specification(int,float,int,int,FILE *,int,int,int *,int *
 ,float,float,int,int,int,int,int,float);
void edit_universes(int,int,int *,int *,int,int,int,int,int,int,int *
 ,int *);
void edit_surfaces(surface_usage_list *);
void editmaterials(usage_list *);
void edit_spacer(int,int *,float *,ascii_string *);
void spacer_location(int,int,float,int *,float *,int *);
```

```
augmented_lattice_list *memory_lattice_list(int
 ,augmented_lattice_list *);
void augment_lattice_list(int *,int *,int *,int,int,int,int,int,int *
 ,int *,augmented_lattice_list *,int *);
ll_material *material_match(a_record *,char[],float *,int *);
void rollup_llm(ll_material *);

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Global Variables - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Integer Variables
 - -      nline - number of lines on current output page
 - -    version - version number of code
 - -  storage_i - current integer storage requested (bytes)
 - -  storage_r - current real storage requested (bytes)
 - -  storage_c - current character storage requested (bytes)
 - - storage_it - maximum integer storage requested (bytes)
 - - storage_rt - maximum real storage requested (bytes)
 - - storage_ct - maximum character storage requested (bytes)
 - -      cdate - date of code execution
 - -     crtime - time of code execution
 - -        pid - process identifier for code execution
*/
  short nline = 0, version = 0;
  int storage_i = 0, storage_r = 0, storage_it = 0, storage_rt = 0,
      storage_c = 0, storage_ct = 0;
  int long pid;
/* - Character Variables */
  char modification_level = '-';
  char codenm[8] = "BLINK";
  char cdate[9] = {NULL};
  char crtime[9] = {NULL};
  ascii_string case_title;
/* - FILE Pointers */
  FILE *nout;

void main(int argc, char *argv[]) {
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  B W R L I N K  *  Creates MCNP Input Decks for Modeling of
 - -                      Commercial Reactor Critical "Experiments"
 - -                      (CRC) with Fuel Constituent Number Densities
 - -                      from SAS2H Analyses
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Command Line Argument(s):
 - -   prefix - Prefix for Names of Input and Output Files
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         nin - FORTRAN logical unit number for input file
 - -     lucgeom - FORTRAN logical unit number for core geometry dataset
 - -               file
 - -  length_fin - length of input file name
```

```
- -      naxial - number of axial nodes in fuel assemblies
- -        nrow - number of rows in core map for the whole core
- -        ncol - number of columns in core map for the whole core
- -       nrowp - number of rows in core map in problem
- -       ncolp - number of columns in the core map in the problem
- -       nrowb - number of control blade location rows for the whole
- -               core
- -       ncolb - number of control blade location columns for the
- -               whole core
- -      nrowbp - number of control blade rows in the problem
- -      ncolbp - number of control blade columnes in the problem
- -     nbundlg - number of unique fuel assemblies geometrical indices
- -     nbundlm - number of unique fuel assemblies material indices
- -    nlatticg - number of unique fuel lattice geometrical indices
- -    nlatticm - number of unique fuel lattice material indices
- -       naxpl - naxial+1
- -       valid - map of valid fuel locations in core (dynamic)
- -        gmap - map of fuel assembly geometrical indices (dynamic)
- -        mmap - map of fuel assembly geometrical indices (dynamic)
- -      lgvect - vector for lattice geometrical assignment to fuel
- -               lattices
- -      lmvect - vector for lattice material assigment to fuel
- -               lattices
- -      validb - map of valid control blade locations in core (dynamic)
- -      bladep - map of control blade axial positions (dynamic)
- -  incore_loc - map of locations of incore instrumentation locations
- -               (dynamic)
- -          lt - scratch integer array used for input processing
- -               (dynamic)
- -       nsrck - nominal source size per cycle in MCNP
- -         ikz - number of cycles to be skipped before beginning tally
- -               accumulation in MCNP
- -         kct - number of cycles to be performed in MCNP
- -       ncell - counter for number of cells generated for non-trans-
- -               lated cells (this begins @ 2001 to give headroom
- -               for indexing of cells that must be translated)
- -    nsurface - counter for number of surfaces generated
- -   nmaterial - counter for number of materials generated
- -      core_f - fraction of core in problem
- -               (1 - full, 2 - half, 4 - quarter)
- -   nuniverse - number of universes in problem
- -      nrowcc - number of rows in control cell map
- -      ncolcc - number of columns in control cell map
- -       ccmap - pointer to map of unique control cells
- -       ntube - number of absorber tubes in control blade
- -         ncs - number of central stiffeners in control blade wing
- - ptr_correspondence_table
- -             - pointer to table containing array giving correspondence
- -               between fuel geometry indices and fuel material indices
- -     ptr_ufl - pointer to vector of universe indices corresponding to
- -               the lattice models created for each unique lattice type
- -     ptr_ufa - pointer to vector of universe indices corresponding to
- -               the fuel assembly models created for each unique
- -               fuel assembly type
```

```
 - -          ucb - universe of control blade
 - -     fau_fill - pointer to array containing fuel assembly assignments
 - -                to control cells
 - -          ndm - periodicity for dumps to TPE file
 - -          mct - flag for creating MCTAL file
 - -                (0 - no file, <> 0 - create file)
 - -         ndmp - maximum number of dumps
 - - ptr_n_spacer
 - -              - pointer to vector containing number of spacers for
 - -                each fuel assembly geometrical type
 - - ptr_spacer_node
 - -              - pointer to array containing the axial node locations
 - -                for each fuel assembly geometrical type
 - - total_spacer_locations
 - -              - total number of spacer locations in geometrically
 - -                unique fuel assemblies
 - - nlatticm_ref
 - -              - number of lattice material indices before addition of
 - -                indices for spacer grid treatment
 - -     ncell_tr - cell indicies for translated cells
*/
  int nin = 5, lucgeom = 1, nsrck = 10000, ikz = 10, kct = 310;
  int ndm = 5, mct = 1, ndmp = 2;
  int ncell = 2000, nsurface = 0, nmaterial = 0, core_f = 4;
  int nuniverse = 0, ncell_tr = 0;
  int ntube = 0, ncs = 0, ucb, nlatticm_ref;
  int nrowcc, ncolcc, total_spacer_locations;
  int length_fin = 132;
  int naxial, nrow, ncol, nrowp, ncolp, nrowb, ncolb, nrowbp, ncolbp
    , nbundlg, nbundlm, nlatticg, nlatticm,naxpl;
  int *valid, *gmap, *mmap, *lgvect, *lmvect, *validb, *bladep, *incore_loc
  , *lt;
  int *ccmap, *ptr_n_spacer, *ptr_spacer_node;
  int *ptr_correspondence_table, *ptr_ufl, *ptr_ufa, *fau_fill;
/* - Float Variable(s)
 - -                afl - active fuel length
 - -              apitch - fuel assembly pitch
 - -                sod - shroud outer radius
 - -              sthick - shroud thickness
 - -                vod - pressure vessel outer radius
 - -              vthick - pressure vessle thickness
 - -               tutpr - Top of Upper Tie Plate Region
 - -                tcgr - Top of Core Grid Region
 - -               bltpr - Bottom of Lower Tie Plate Region
 - -                bcpr - Bottom of Core Plate Region
 - -                rkk - initial guess for eigenvalue in MCNP
 - -    bypass_density - bypass water density
 - - inchannel_density - in-channel water density
 - - blade_window_offset
 - -                   - sizing of windows for control blades
 - -                     and fuel assemblies in control cell
 - -                     template
 - -             cbspan - control blade span
 - -               atid - blade absorber tube inner diameter (cm)
```

```
 - -              atod - blade absorber tube outer diameter (cm)
 - -            cbthick - blade wing thickness (cm)
 - -             trspan - blade tie rod span (cm)
 - -            trthick - blade tie rod thickness (cm)
 - -           cblength - blade active absorber length (cm)
 - -            wsthick - blade wing thickness (cm)
 - -              csoff - blade central stiffener offset (cm)
 - -            cswidth - blade central stiffener width (cm)
 - -               dtod - incore guide tube outer diameter (cm)
 - -               dtid - incore guide tube inner diameter (cm)
 - -            ptr_nab - axial nodal boundary coordinates
 - -          cb_stroke - control blade stroke (cm)
 - -                rho - input in-channel density value (g/cc)
 - -             rhobyp - input bypass density value (g/cc)
 - -              tempk - input problem temperature (Kelvins)
 - - ptr_spacer_location
 - -                    - pointer to array containing spacer
 - -                      axial location (w/r/t Bottom of Active
 - -                      Fuel)
*/
  float afl, apitch, sod, sthick, vod, vthick, tutpr, tcgr, bltpr
  , bcpr, dtod, dtid, rho = 0.0, rhobyp = 0.0, tempk = 293.15;
  float cbspan = 0.0, atid = 0.0, atod = 0.0, cbthick = 0.0
   ,trspan = 0.0, trthick = 0.0, cblength = 0.0, wsthick = 0.0
   ,csoff = 0.0, cswidth = 0.0, cb_stroke = 365.76;
  float rkk = 1.0, bypass_density = 1.0, inchannel_density = 1.0;
  float blade_window_offset = 0.40;
  float *ptr_nab, *ptr_spacer_location;
/* - Character Variables
 - -             fin - name for input file
 - -            fout - name for output file
 - -          inname - FORTRAN NAMELIST on input file
 - -          prefix - prefix for names of input and output files
 - -         core_db - name of core geometry database file
 - -         lprefix - prefix for lattice geometry database
 - -         fprefix - prefix for fuel material intermediate database
 - -         lgdsnam - pointer to names of datasets for unique geometrical
 - -                   lattice types
 - -         lmdsnam - pointer to names of datasets for unique geometrical
 - -                   lattice types
 - -       cell_file - name for scratch file for cell definitions
 - -    surface_file - name for scratch file for surface definitions
 - - material_file - name for scratch file for material definitions
 - -       MCNP_file - name for MCNP input file
 - -       core_mtls - name of core materials database file
 - -         mvessel - vessel material identifier
 - -         mshroud - core shroud material identifier
 - -             mtg - core top guide region material identifier
 - -             mcp - core plate region material identifier
 - -            mutp - upper tie plate region material identifier
 - -            mltp - lower tie plate region material identifier
 - -            migt - incore guide tube material identifier
 - -        blade_db - dataset containing blade geometry for problem
 - -          cbpmat - identifier for blade poison material
```

```
 - -          atmat - identifier for absorber tube poison material
 - -         cbsmat - identifier for blade sheath material
 - -        cbtrmat - identifier for blade tie rod material
 - - ptr_spacer_material
 - -                - pointer to vector containing alphanumeric
 - -                  identifiers for each fuel assembly spacer
 - -                  material mnemonic
*/
  char fin[133], fout[133];
  char inname[8] = "LINKIN";
  char prefix[133], core_db[133], lprefix[133], fprefix[133]
   , cell_file[133], surface_file[133], material_file[133]
   , MCNP_file[133], core_mtls[133], blade_db[133];
  char mshroud[6], mvessel[6], mtg[6], mcp[6], mutp[6], mltp[6]
   , migt[6];
  char cbpmat[6], atmat[6], cbsmat[6], cbtrmat[6];
  ascii_string *lgdsnam, *lmdsnam, *ptr_spacer_material;
/* - file pointers
 - -   lmcnp - file for MCNP Input Deck
 - -     lu8 - scratch file for MCNP card images for cell definitions
 - -     lu9 - scratch file for MCNP card images for surface definitions
 - -    lu10 - scratch file for MCNP card images for material defini-
               tions
*/
  FILE *lmcnp, *lu8, *lu9, *lu10;
/* - Structured Variable(s)
 - -       ptr_core_mtls - pointer to first record of linked list for
contents
 - -                       of core materials dataset
 - -   ptr_surface_usage - pointer to first record of linked list for
 - -                       surface labels and indices
 - -  ptr_material_usage - pointer to first record of linked list for
 - -                       material labels and indices
 - -           ptr_lg_ds - pointer to lattice geometry datasets
 - - additional_lattices - material lattice indices for lattices added for
 - -                       spacer grid treatment
*/
  a_record *ptr_core_mtls;
  usage_list *ptr_material_usage;
  surface_usage_list *ptr_surface_usage;
  fg_list *ptr_lg_ds;
  augmented_lattice_list *additional_lattices;
/* - initialize global variables not explicitly initialized elsewhere */
/* - verify presence of prefix for input and output file names, if found
 - - construct files names - - - - - - - - - - - - - - - - - - - - - - - */
  if(argc < 2) {
    puts("No file prefix available from command line -- abort\n");
    return;}
  else {
    strcpy(prefix,argv[1]);
    strcpy(fin,prefix);
    strcat(fin,".inp");
    strcpy(fout,prefix);
    strcat(fout,".out");
```

```
  }
/* - open input and output files for processing */
  { FILE *inp;
    inp = fopen(fin,"r");
    if(!inp) {
      puts("Input file cannot be opened\n");
      return;}
    fclose(inp);
  }
  nout = fopen(fout,"w");
  header();
/* - echo input stream to output stream */
  echo(inname,fin);
/* - Process Input Directives - - - - - - - - - - - - - - - - - - - - - - */
/* - Read Namelists */
  readin(fin,&length_fin,core_db,&nin,&lucgeom,&naxial,&afl,&nrow
   ,&ncol,&nrowp,&ncolp,&apitch,&sod,&sthick,&vod,&vthick
   ,&tutpr,&tcgr,&bltpr,&bcpr,lprefix,fprefix,core_mtls,mvessel
   ,mshroud,mtg,mcp,mutp,mltp,&core_f,blade_db,migt,&dtod,&dtid
   ,&rho,&rhobyp,&tempk);
/* - Set interal variables for in-channel and bypass water densities */
  if(rho != 0.0) inchannel_density = rho;
  if(rhobyp != 0)
    bypass_density = rhobyp;
  else
    if(rho != 0)
      bypass_density = rho;
/* - Allocate Memory for Arrays */
  nrow%2?(nrowb = nrow/2+1):(nrowb = nrow/2);
  ncol%2?(ncolb = ncol/2+1):(ncolb = ncol/2);
  nrowp%2?(nrowbp = nrowp/2+1):(nrowbp = nrowp/2);
  ncolp%2?(ncolbp = ncolp/2+1):(ncolbp = ncolp/2);
  valid = memory_integer(1,(nrow*ncol),valid);
  gmap = memory_integer(1,(nrowp*ncolp),gmap);
  mmap = memory_integer(1,(nrowp*ncolp),mmap);
  validb = memory_integer(1,(nrowb*ncolb),validb);
  bladep = memory_integer(1,(nrowbp*ncolbp),bladep);
  incore_loc = memory_integer(1,(nrowb*ncolb),incore_loc);
  lt = memory_integer(1,(nrowp*ncolp),lt);
/* - Read Arrays from Input and Core Geometry File */
  { int maxb;
    maxb = nrowp*ncolp;
    redmaps(&nin,&lucgeom,&nrow,&ncol,&nrowp,&ncolp,valid,gmap,validb
     ,bladep,&nbundlg,&maxb,lt,&nrowb,&ncolb,&nrowbp,&ncolbp
     ,incore_loc,mmap,&nbundlm);
  }
/* - Return Scratch Memory and Arrays that are No Longer Needed - - */
  valid = memory_integer(-1,(nrow*ncol),valid);
  validb = memory_integer(-1,(nrowb*ncolb),validb);
  lt = memory_integer(-1,(nrowp*ncolp),lt);
/* - Read Lattice Assignments to Fuel Types - - - - - - - - - - - - - */
  naxpl = naxial+1;
  lgvect = memory_integer(1,(naxpl*nbundlg),lgvect);
  ldlv(&nbundlg,&naxpl,&nin,lgvect);
```

```
  lmvect = memory_integer(1,(naxpl*nbundlm),lmvect);
  ldlv(&nbundlm,&naxpl,&nin,lmvect);
/* - Determine the Number of Unique Lattices */
  { int max;
    max = naxpl*nbundlg;
    lt = memory_integer(1,max,lt);
    nlatticg = (int) clattice(&naxpl,&nbundlg,&max,lgvect,lt);
    lt = memory_integer(-1,max,lt);
    max = naxpl*nbundlm;
    lt = memory_integer(1,max,lt);
    nlatticm = (int) clattice(&naxpl,&nbundlm,&max,lmvect,lt);
    lt = memory_integer(-1,max,lt);
  }
/* - Read Names of Datasets for Unique Lattice Geometrical and - - -
 - - Material Indices */
/* - Dataset Names are Limited to 132 Characters */
  lgdsnam = memory_ascii_string(1,(nlatticg),lgdsnam);
  lmdsnam = memory_ascii_string(1,(nlatticm),lmdsnam);
/* - Dataset Names for Fuel Geometries */
  { int len = 133;
    strngr(&nin,&len,&nlatticg,lgdsnam);
/* - Dataset Names for Fuel Material Intermediate Datasets */
    strngr(&nin,&len,&nlatticm,lmdsnam);
  }
/* - Spacer Information - -  - - - - - - - - - - - - - - - - - - - - */
  { short int i;
    int one = 1;
/* - Read Number of Spacers for Each Fuel Assembly Geometrical
 - - Type */
    ptr_n_spacer = memory_integer(1,nbundlg,ptr_n_spacer);
    ldlv(&one,&nbundlg,&nin,ptr_n_spacer);
/* - Compute Total Number of Spacers in Problem */
    { short int i;
      int *ptr = ptr_n_spacer;
      for(i = 1;i <= nbundlg;i++){
        total_spacer_locations += *ptr;
        ptr++;}
    }
/* - Read in Spacer Locations and Material Mnemonic for each Fuel
 - - Assembly Geometrical Type */
    ptr_spacer_location = memory_float(1,total_spacer_locations
     ,ptr_spacer_location);
    ptr_spacer_material = memory_ascii_string(1,nbundlg
     ,ptr_spacer_material);
    ldlvr(&one,&total_spacer_locations,&nin,ptr_spacer_location);
    { short int i;
      int len = 133;
      strngr(&nin,&len,&nbundlg,ptr_spacer_material);
    }
  }
/* - Load Contents of Fuel Geometry Datasets - - - - - - - - - - -*/
/* - Allocate Memory */
  ptr_lg_ds = memory_fg_list(1,nlatticg,ptr_lg_ds);
  { short int i;
```

```
    int latdim, nwr;
    float cthick, asin, wgap, ngap, cradius, fsrd, cfsrd, rpitch
     ,cod, cld, pod;
    char frcmat[7], fcmat[7];
    ascii_string *as = lgdsnam, fn;
    fg_list *p;
    p = ptr_lg_ds;
      { int len=132, length;
        length = mchar(&len,lprefix);
        lprefix[length] = '\0';
        length = mchar(&len,fprefix);
        fprefix[length] = '\0';
      }
    for( i = 1; i <= nlatticg; i++,p++,as++){
      { int len=132, length;
        ascii_string holder;
        strncpy(holder,*as,len);
        length = mchar(&len,holder);
        holder[length] = '\0';
        strncpy(*as,holder,(length+1));
      }
      strcpy(fn,lprefix);
      strcat(fn,*as);
      rlattice(&length_fin,fn,&lucgeom,&cthick,&asin,&wgap,&ngap
       ,&cradius,&fsrd,&cfsrd,&rpitch,&cod,&cld,&pod,frcmat,fcmat
       ,&latdim,&nwr);
      strcpy(p->gds_name,*as);
      p->latdim = latdim; p->nwr = nwr; p->cthick = cthick;
      p->asin = asin; p->wgap = wgap; p->ngap = ngap;
      p->cradius = cradius; p->fsrd = fsrd; p->cfsrd = cfsrd;
      p->rpitch = rpitch; p->cod = cod; p->cld = cld;
      p->pod = pod;
      { int len=6, length;
        length = mchar(&len,frcmat);
        frcmat[length] = '\0';
        length = mchar(&len,fcmat);
        fcmat[length] = '\0';
      }
      strncpy(p->frcmat,frcmat,6);
      strncpy(p->fcmat,fcmat,6);}
  }
/* - Close Input File */
  ftclose(&nin);
/* - Compute Nodal Locations of Spacers */
  ptr_spacer_node = memory_integer(1,total_spacer_locations
   ,ptr_spacer_node);
  spacer_location(naxial,nbundlg,afl,ptr_n_spacer
   ,ptr_spacer_location,ptr_spacer_node);
/* - Read Contents of Blade Geometry Dataset */
  rblade(&length_fin,blade_db,&lucgeom,&ntube,&cbspan,&atid,&atod
   ,&cbthick,&trspan,&trthick,&wsthick,&cblength,&ncs,&csoff,&cswidth
   ,cbpmat,atmat,cbsmat,cbtrmat);
/* - Generate Correspondence Table of Fuel Geometry Indices and Fuel
 - - Material Indices */
```

```
   ptr_correspondence_table =
    memory_integer(1,(2*nlatticm),ptr_correspondence_table);
   lodct(&nrowp,&ncolp,&naxpl,&nlatticg,&nlatticm,gmap,mmap
    ,lgvect,lmvect,ptr_correspondence_table);
/* - Edit Input Instructions - - - - - - - - - - - - - - - - - - - - - - - - */
  editin(core_db,lprefix,fprefix,naxial,naxpl,afl,nrowp,ncolp,nrowbp
   ,ncolbp,nbundlg,nbundlm,gmap,mmap,lgvect,lmvect,bladep,nlatticg
   ,nlatticm,lgdsnam,lmdsnam,core_mtls,core_f,blade_db,rho,rhobyp
   ,tempk,mutp,mltp);
/* - Edit Contents of Core Geometry Dataset - - - - - - - - - - - - - - - */
  coredb_edt(nrow,ncol,nrowb,ncolb,apitch,vod,vthick,sod,sthick,tutpr
   ,tcgr,bltpr,bcpr,incore_loc,mvessel,mshroud,mtg,mcp,migt,dtod,dtid);
/* - Edit Contents of Blade Geometry Dataset - - - - - - - - - - - - - - - */
  bladedb_edt(ntube,cbspan,atid,atod,cbthick,trspan,trthick,wsthick
   ,cblength,ncs,csoff,cswidth,cbpmat,atmat,cbsmat,cbtrmat);
/* - Edit Contents of Fuel Geometry Datasets - - - - - - - - - - - - - - - */
  fgds_edt(nlatticg,lprefix,lgdsnam,ptr_lg_ds);
/* - Edit Spacer Input Information - - - - - - - - - - - - - - - - - - - - - */
  edit_spacer(nbundlg,ptr_n_spacer,ptr_spacer_location
   ,ptr_spacer_material);
/* - Edit Fuel Material/Geometry Correspondence Table - - - - - - - - - - - */
  edit_ct(nlatticm,ptr_correspondence_table);
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Add Lattice Definitions for Spacer Grid Treatment - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  nlatticm_ref = nlatticm;
  additional_lattices = memory_lattice_list(1,additional_lattices);
  augment_lattice_list(&nlatticm,lmvect,lgvect,nlatticm_ref,nlatticg
   ,nbundlm,nbundlg,naxial,ptr_n_spacer,ptr_spacer_node
   ,additional_lattices,ptr_correspondence_table);
/* - Regenerate Correspondence Table */
  ptr_correspondence_table = memory_integer(-1,(2*nlatticm_ref)
   ,ptr_correspondence_table);
  ptr_correspondence_table = memory_integer(1,(2*nlatticm)
   ,ptr_correspondence_table);
  lodct(&nrowp,&ncolp,&naxpl,&nlatticg,&nlatticm,gmap,mmap,lgvect
   ,lmvect,ptr_correspondence_table);
/* - Edit New Lattice Material Index Vector and Correspondence Table */
  lines(7);
  fprintf(nout,"0Lattice Material Index Vector has been Regenerated");
  fprintf(nout," to Accomodate Spacer Modeling\n");
  fprintf(nout,"0Fuel Assembly Index/Lattice Index\n");
  { short int i,j;
    int *p = lmvect;
    fprintf(nout,"bundle");
    for(i = 1;i <= naxial;i++)
      fprintf(nout,"%5i",i);
    fprintf(nout,"\n\n");
    for(j = 0;j < nbundlm;j++){
      fprintf(nout,"%5i ",*p);
      p++;
      for(i = 1;i < naxpl;i++){
        fprintf(nout,"%5i",*p);
        p++;
```

```
      }
      if(i > 1) lines(1);
      fprintf(nout,"\n");
    }
  }
  lines(2);
  fprintf(nout,"0New Correspondence Table\n\n");
  edit_ct(nlatticm,ptr_correspondence_table);
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Generate Input Deck for MCNP - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  strcpy(cell_file,prefix);
  strcat(cell_file,".cel");
  strcpy(surface_file,prefix);
  strcat(surface_file,".sur");
  strcpy(material_file,prefix);
  strcat(material_file,".mat");
/* - Open Scratch Files for Segments of MCNP Input Deck */
  lu8 = fopen(cell_file,"w");
  lu9 = fopen(surface_file,"w");
  lu10 = fopen(material_file,"w");
/* - Load Core Materials Database into Memory */
  ptr_core_mtls = load_core_mtls(core_mtls,length_fin);
/* - Set up first acess to surface usage pointer */
  ptr_surface_usage =
   memory_surface_usage_list(1,ptr_surface_usage);
  ptr_surface_usage->last = NULL;
  ptr_surface_usage->index = 0;
  strcpy("",ptr_surface_usage->label);
  strcpy("",ptr_surface_usage->value);
  strcpy("",ptr_surface_usage->mnemonic);
  strcpy("",ptr_surface_usage->equivalent_label);
  ptr_surface_usage->next = NULL;
/* - Write Representations for Core Structural Components */
  vessel_generation(apitch,vod,vthick,sod,sthick,tutpr,tcgr,bltpr,bcpr
   ,lu8,lu9,lu10,&ncell,&nsurface,&nmaterial,ptr_core_mtls,mvessel
   ,mshroud,mtg,mcp,mutp,mltp,core_f,ptr_surface_usage
   ,&ptr_material_usage,bypass_density,afl);
/* - Generate Control Blade Model */
  build_control_blade(ntube,cbspan,atid,atod,cbthick,trspan,trthick
   ,wsthick,cblength,ncs,csoff,cswidth,cbpmat,atmat,cbsmat,cbtrmat
   ,ptr_material_usage,ptr_surface_usage,ptr_core_mtls,lu8,lu9,lu10
   ,&nuniverse,&nmaterial,bypass_density,&ncell,&ucb,&nsurface
   ,&ncell_tr);
/* - Create Fuel Assembly Models - - - - - - - - - - - - - - - - - - - - - - - */
/* - Create In-channel Moderator Material */
  { short int i, nloc;
    char *cp, zaid[11];
    ascii_string label;
    usage_list *ptr_ml = ptr_material_usage;
    do{
      ptr_ml = ptr_ml->next;}
    while(ptr_ml->next != NULL);
    strcpy(label,"Inchannel Water");
```

```
    fprintf(lu10,"c       %s\n",label);
    nmaterial++;
    ptr_ml = load_usage_list(label,nmaterial,ptr_ml);
    strcpy(zaid,"001001.50c");
    nloc = 0;
    if(nmaterial < 1000) nloc++;
    if(nmaterial < 100) nloc++;
    if(nmaterial < 10) nloc++;
    strcpy(label,"2.0");
    { short int ip;
      for( ip = 1; ip <= nloc; ip++) fprintf(lu10," ");
      fprintf(lu10,"m%i %s %s\n",nmaterial,zaid,label);
    }
    strcpy(zaid,"008016.50c");
    strcpy(label,"1.0");
    fprintf(lu10,"       %s %s\n",zaid,label);
    { short int ip;
      for( ip = 1; ip <= (nloc-1); ip++) fprintf(lu10," ");
      fprintf(lu10,"mt%i lwtr.01\n",nmaterial);
    }
  }
  ptr_ufl = memory_integer(1,nlatticm,ptr_ufl);
  { int n, ufl, nlat, index, n_entries;
    float density;
    int *p_ufl = ptr_ufl;
    ascii_string *p_sm = ptr_spacer_material;
    ll_material *ptr_mtl;
    usage_list *ptr_ml;
    augmented_lattice_list *p_addlat = additional_lattices;
    for(n = 1; n <= nlatticm_ref; n++,p_ufl++){
      generate_lattice_model(&ncell,&nmaterial,lu8,lu9,lu10
       ,ptr_surface_usage,ptr_material_usage
       ,ptr_correspondence_table,ptr_lg_ds,lmdsnam,n,fprefix
       ,inchannel_density,bypass_density,&nuniverse
       ,ptr_core_mtls,&ufl,&nsurface,"Inchannel Water");
      *p_ufl = ufl;}
/* - Add Lattices Incorporating Spacer Grids */
    for(n = (nlatticm_ref+1);n <= nlatticm;n++,p_ufl++){
/* - Determine Material Density and Add Composition to Input
 - - Deck */
      { short int i;
        int len = 132, length, *lmv = lmvect, nlatm, nlatg;
        int *p = ptr_correspondence_table;
        ascii_string buffer;
/* - Find Lattice Geometry Index Corresponding to Lattice
 - - Material Index */
        for(i = 1;i < n;i++) lmv++;
        nlatm = *lmv;
        for(i = 1;i < nlatm;i++) {p++; p++;}
        p++;
        nlatg = *p;
        for(i = 1;i < nlatg;i++,p_sm++);
        strncpy(buffer,*p_sm,len);
        length = mchar(&len,*p_sm);
```

```
        buffer[length] = '\0';
        strncpy(*p_sm,buffer,len);
      }
      ptr_mtl = material_match(ptr_core_mtls,*p_sm,&density
       ,&n_entries);
      search_usage_list(1,*p_sm,&index,ptr_material_usage);
      if(index == 0){
        nmaterial++;
        ptr_ml = ptr_material_usage;
        while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
        index = (ptr_ml->index)+1;
        ptr_ml = load_usage_list(*p_sm,index,ptr_ml);
        add_material(lu10,&nmaterial,*p_sm,n_entries,ptr_mtl);
        { short int nb,nblank = 0;
          if(nmaterial < 100) nblank = 1;
          if(nmaterial < 10) nblank = 2;
          for(nb = 1;nb <= nblank;nb++) fprintf(lu10," ");
          fprintf(lu10,"mt%i lwtr.01\n",nmaterial);
        }
        rollup_llm(ptr_mtl);}
/* - Generate New Lattices */
      nlat = p_addlat->basis_lattice_material_index;
      generate_lattice_model(&ncell,&nmaterial,lu8,lu9,lu10
       ,ptr_surface_usage,ptr_material_usage
       ,ptr_correspondence_table,ptr_lg_ds,lmdsnam,nlat
       ,fprefix,density,bypass_density,&nuniverse
       ,ptr_core_mtls,&ufl,&nsurface,*p_sm);
      *p_ufl = ufl;
      p_addlat = p_addlat->next;
    }
  }
/* - Compute Nodal Axial Boundaries */
  ptr_nab = memory_float(1,(naxial-1),ptr_nab);
  { short int i;
    float delta, last, *p=ptr_nab;
    delta = afl/((float) naxial);
    last = 0.0;
    for(i = 1;i <= (naxial-1);i++,p++){
      *p = last+delta;
      last = *p;
    }
  }
/* - Generate Unique Lattice Definitions - - - - - - - - - - - - */
  ptr_ufa = memory_integer(1,nbundlm,ptr_ufa);
  build_assemblies(&ncell,&nuniverse,lu8,lu9,ptr_surface_usage
   ,ptr_nab,&ptr_ufa,nbundlm,lmvect,naxial,ptr_ufl,&nsurface);
  ptr_nab = memory_float(-1,(naxial-1),ptr_nab);
/* - Determine the Location and Number of Unique Control Cells */
  { int *ccmapw, nu_cc;
    nu_cc = nuniverse+1;
    if(core_f == 4){
      ncolcc = (ncolp/2)+1;
      nrowcc = (nrowp/2)+1;}
    if(core_f == 1){
```

```
      nrowp%4?(nrowcc = nrowp/2):(nrowcc = (nrowp/2)+1);
      ncolp%4?(ncolcc = ncolp/2):(ncolcc = (ncolp/2)+1);}
    if(core_f == 2){
      ncolcc = (ncolp/2)+1;
      nrowp%4?(nrowcc = nrowp/2):(nrowcc = (nrowp/2)+1);}
    ccmap = memory_integer(1,(ncolcc*nrowcc),ccmap);
    ccmapw = memory_integer(1,(6*ncolcc*nrowcc),ccmapw);
    ccmgen(&nrowp,&ncolp,mmap,&nrowcc,&ncolcc,ccmap,ccmapw,&core_f
     ,&ncolbp,&nrowbp,bladep,&nuniverse,incore_loc,&nrowb,&ncolb);
/* - Generate Control Cells */
    fau_fill = memory_integer(1,4*(nuniverse-nu_cc+1),fau_fill);
    build_control_cells(&ncell_tr,&nuniverse,ncolcc,nrowcc,ccmap,ccmapw
     ,(nuniverse-nu_cc+1),lu8,lu9,blade_window_offset,ptr_surface_usage
     ,apitch,ucb,cbspan,cb_stroke,migt,ptr_material_usage,ptr_core_mtls
     ,&nmaterial,ptr_ufa,lu10,bypass_density,dtid,dtod,nu_cc,fau_fill
     ,&nsurface);
/* - Genrate Control Cell Lattice */
    core_lattice_generation(&ncell,&nsurface,apitch,bypass_density
     ,ptr_surface_usage,ptr_material_usage,ccmap,ncolcc,nrowcc
     ,lu8,lu9,core_f,&nuniverse,afl);
    edit_universes(ncolcc,nrowcc,ccmap,fau_fill,core_f,ncolp,nrowp
     ,nu_cc,nlatticm,nbundlm,ptr_ufl,ptr_ufa);
    ptr_ufl = memory_integer(-1,nlatticm,ptr_ufl);
    source_specification(nsrck,rkk,ikz,kct,lu10,ncolcc,nrowcc,ccmap
     ,fau_fill,apitch,afl,core_f,ndm,mct,ndmp,(ncell-2000+ncell_tr)
     ,tempk);
    fau_fill = memory_integer(-1,4*(nuniverse-nu_cc+1),fau_fill);
    ccmapw = memory_integer(-1,(6*ncolcc*nrowcc),ccmapw);
  }
  ptr_ufa = memory_integer(-1,nbundlm,ptr_ufa);
/* - Edit Surfaces Generated */
  edit_surfaces(ptr_surface_usage);
  edit_materials(ptr_material_usage);
/* - Close Scratch File for Deck Segments */
  fclose(lu8);
  fclose(lu9);
  fclose(lu10);
/* - Build MCNP Deck from Segment Files */
  strcpy(MCNP_file,prefix);
  strcat(MCNP_file,"_m.inp");
  lmcnp = fopen(MCNP_file,"w");
/* - Open Scratch Files for Segments of MCNP Input Deck */
  lu8 = fopen(cell_file,"r");
  lu9 = fopen(surface_file,"r");
  lu10 = fopen(material_file,"r");
  generate_deck(lmcnp,lu8,lu9,lu10);
  fclose(lu8); fclose(lu9); fclose(lu10);
  fclose(lmcnp);
/* - Discard Scratch Files */
  discard_scratch_file(cell_file);
  discard_scratch_file(surface_file);
  discard_scratch_file(material_file);
/* - Echo MCNP Input Deck to Output File */
  lmcnp = fopen(MCNP_file,"r");
```

```
  echo_MCNP_deck(lmcnp);
/* - Return Memory */
  incore_loc = memory_integer(-1,(nrowb*ncolb),incore_loc);
  bladep = memory_integer(-1,(nrowbp*ncolbp),bladep);
  gmap = memory_integer(-1,(nrowp*ncolp),gmap);
  mmap = memory_integer(-1,(nrowp*ncolp),mmap);
  lgvect = memory_integer(-1,(naxpl*nbundlg),lgvect);
  lmvect = memory_integer(-1,(naxpl*nbundlm),lmvect);
  lgdsnam = memory_ascii_string(-1,(nlatticg),lgdsnam);
  lmdsnam = memory_ascii_string(-1,(nlatticm_ref),lmdsnam);
  ptr_lg_ds = memory_fg_list(-1,nlatticg,ptr_lg_ds);
   ptr_correspondence_table =
    memory_integer(-1,(2*nlatticm),ptr_correspondence_table);
/* - end of processing - - - - - - - - - - - - - - - - - - - - - - - - - - */
  memsum();
  fclose(nout);
  return;
}
```

2.2. Service Routines

Function abort

```
#include <stdio.h>

void abort(){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  a b o r t  * Handles abnormal terminations detected by the
 - -                  calling routine
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - -
 - - FILE Variable(s)
 - -  nout - pointer to code output file
*/
  extern FILE *nout;
/* - Write Termination Processing Message - - - - - - - - - - - - - - - - */
  fprintf(nout,"0*** Processing Stops  ***\n");
  exit(0);
  return;
}
```

Function bufferpad

```
void bufferpad(char buffer[], int start, int last){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * b u f f e r p a d  *  Adds blanks to character string
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     buffer - string variable to which add blanks          (input)
 - -      start - starting location for adding blanks          (input)
 - -       last - ending location for adding blanks            (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  i - counter
*/
  short int i, begin;
/* - Character Variable(s)
 - -  blank - blank character */
  char blank = ' ';
  if(start == 0){
    strncpy(buffer,"",1);
    begin = 1;}
  else begin = start;
  for( i = begin; i <= last; i++)
    strncat(buffer,&blank,1);
}
```

Function copy_ascii_file

```
#include<stdio.h>
#include<string.h>

void copy_ascii_file(FILE *lu1,FILE *lu2){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * c o p y _ a s c i i _ f i l e  *  Copies Contents of One ASCII
 - -                                      File to Another
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -        lu1 - pointer to target file                          (input)
 - -        lu2 - pointer to source file                          (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Character Variable(s)
 - -       c - single character used to read from source file
 - -       p - pinter ot character variable
 - -  buffer - string variable used to load entire lines for print
*/
  unsigned char c;
  char *p;
  char buffer[81];
/* - Copy Line by Line - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  do
  { strcpy(buffer,"");
    p = buffer;
    do
    { c = fgetc(lu2);
      *p = c;
      p++;}
    while((c != 10) && (c != 255));
    if(c != 255){
      p = buffer;
      c = *p;
      while(c != 10){
        fputc(*p,lu1);
        p++;
        c = *p;}
      fprintf(lu1,"\n");}
  } while(c != 255);
  return;
}
```

Function discard_scratch_file

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

void discard_scratch_file(char filenm[]){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * d i s c a r d _ s c r a t c h _ f i l e  *  Creates sub-pro-
 - -                                                to Delete Files
 - -                                                used for Scratch
 - -                                                Processing
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     filenm - specification for file to be discarded       (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - -  Integer Variable(s)
 - -   return_code - return code from system call
*/
 int return_code = 0;
/* -  Character Variable(s)
 - -  buffer - string variable used to load command for sub-process
*/
  char buffer[133];
/* - FILE Variable(s)
 - -  nout - output file string pointer
*/
  extern FILE *nout;
/* - Create Command String for Sub-process - - - - - - - - - - - - - - - - */
  sprintf(buffer,"rm -f %s\n",filenm);
  return_code = system(buffer);
  if(return_code != 0){
    lines(5);
    fprintf(nout,"0***  W a r n i n g  ***  Function");
    fprintf(nout," discard_scratch_file -- Error on rm Call\n");
    fprintf(nout,"0Error Number %i Detected\n");
    fprintf(nout,"\n");
  }
  return;
}
```

Function fortran_message

```
#include <stdio.h>

void fortran_message(char message[132]){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * f o r t r a n _ m e s s a g e * writes messages from FORTRAN
 - -                                     routines to output files
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - .
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - -
 - - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
/* - Write Message to Output File - - - - - - - - - - - - - - - - - - */
  fprintf(nout,message);
  fprintf(nout,"\n");
  return;
}
```

Subroutine ftclose

```
      SUBROUTINE FTCLOSE(UNIT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  F T C L O S E  *  Closes Sequential Files
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -        UNIT - Logical Unit Number Associated with the     (input)
C - -               File to be Closed
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MESSAGE - String for Messages
      INTEGER UNIT
      CHARACTER*132 MESSAGE
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      CLOSE(UNIT=UNIT,STATUS='KEEP',IOSTAT=IOS,ERR=500)
C - - Normal End of Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 CALL lines(2)
      MESSAGE =
     2 '0***  F A T A L  E R R O R  ***  SUBR. FCLOSE -- Error Number'
      LOC = MCHAR(132,MESSAGE)+1
      WRITE(MESSAGE(LOC:(LOC+3)),'(I4)') IOS
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Encountered Closing Loical Unit: '')')
      LOC = MCHAR(132,MESSAGE)+1
      WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') UNIT
      CALL fortran_message(MESSAGE)
      CALL lines(2)
      CALL abort()
      RETURN
      END
```

Subroutine ftopen

```
      SUBROUTINE FTOPEN(UNIT,FNAME)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  F T O P E N  *  Opens Sequential Files for Processing
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -        UNIT - Logical Unit Number to Associate with the   (input)
C - -               File to be Opened
C - -       FNAME - Name of File to Open                        (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MESSAGE - message to pass to output file
      INTEGER UNIT
      CHARACTER*132 MESSAGE
      CHARACTER*(*) FNAME
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      OPEN(UNIT=UNIT,NAME=FNAME,IOSTAT=IOS,ERR=500)
C - - Normal End of Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 CALL lines(3)
      MESSAGE =
     2 '0***  F A T A L  E R R O R  ***  SUBR. FTOPEN -- '
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'(''Error Number '')')
      LOC = MCHAR(132,MESSAGE)+1
      WRITE(MESSAGE(LOC:(LOC+3)),'(I4)') IOS
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Encounterd Opening File:'')')
      CALL fortran_message(MESSAGE)
      MESSAGE = ' '
      WRITE(MESSAGE,'(A)') FNAME
      CALL fortran_message(MESSAGE)
      CALL lines(2)
      CALL abort()
      END
```

Function header

```
#include <stdio.h>
#include <string.h>
#include <time.h>
typedef char ascii_string[133];

void header(){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * h e a d e r * Prints the header at the top of each page
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declaration(s) - - - - - - - - - - - - - - - - - - - - */
/* - Integer Variables
 - -   npage - page number
 - -     pid - process id for linkage code execution
 - -   nline - current line number on output page
 - - version - version number for code
*/
  static short npage = 0;
  extern long pid;
  extern short nline;
  extern short version;
/* - Character Variables
 - -             cdate - date of execution
 - -            crtime - time of execution
 - -            codenm - name of code
 - - modification_level - modification level for code
 - -        case_title - title for this analysis
*/
  extern char cdate[9];
  extern char crtime[9];
  extern char modification_level;
  extern char codenm[5];
  extern ascii_string case_title;
/* - FILE Variable(s)
 - -   nin - input file
 - -  nout - output file
 */
  extern FILE *nin, *nout;
/* - Structures
*/
  struct tm *timeptr;
/* - Initialization for first call - - - - - - - - - - - - - - - - - - */
  if(strlen(cdate) == NULL)
  { size_t maxsize = 9;
    long tloc = {NULL};
    time(&tloc);
    timeptr = localtime(&tloc);
    strftime(cdate,maxsize,"%x",timeptr);
    strftime(crtime,maxsize,"%X",timeptr);
    pid = getpid();
  }
/* - Print FORTRAN-type page eject and header record - - - - - - - - - */
```

```
  npage++;
  fprintf(nout,"1");
  fprintf(nout,"*** ");
  { register unsigned int i;
    char *p;
    p = codenm;
    for (i=0; i < strlen(codenm); i++,p++)
      fprintf(nout,"%c ",*p);
  }
  fprintf(nout,"*** VERS. ");
  fprintf(nout,"%2i",version);
  fprintf(nout,":");
  fprintf(nout,"%c",modification_level);
  fprintf(nout," ** CRWMS,M&O ** DATE: ");
  fprintf(nout,cdate);
  fprintf(nout,", TIME: %s",crtime);
  fprintf(nout," PID: %5i",pid);
  { register unsigned int i;
    for(i = 1;i <= 37;i++) fprintf(nout," ");
  }
  fprintf(nout," PAGE: %u \n",npage);
  if(npage != 1) {
    fprintf(nout," ");
    fprintf(nout,case_title);
    fprintf(nout,"\n");
    fprintf(nout,"\n");
    nline = 4;}
  else
    nline = 3;
  return;
}
```

Subroutine Invali

```
      SUBROUTINE INVALI(NVAL,VECTOR,VALUE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  I N V A L I *  Initializes a integer vector to a single value
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -        NVAL - Number of entries in vector                (input)
C - -      VECTOR - Vector to be initialized                  (output)
C - -       VALUE - Value to initialize vector with            (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER VALUE
      INTEGER VECTOR(NVAL)
C - - Sweep through vector initializing entries - - - - - - - - - - - -
      DO N = 1,NVAL
        VECTOR(N) = VALUE
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine invalr

```
      SUBROUTINE INVALR(NVAL,VECTOR,VALUE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  I N V A L R *  Initializes a real vector to a single value
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -        NVAL - Number of entries in vector                (input)
C - -      VECTOR - Vector to be initialized                  (output)
C - -       VALUE - Value to initialize vector with            (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION VECTOR(NVAL)
C - - Sweep through vector initializing entries - - - - - - - - - - - -
      DO N = 1,NVAL
        VECTOR(N) = VALUE
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Function lines

```
void lines(int nl){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  l i n e s  * Computes the number of cumulative lines printed
 - -                  and determines if a page eject is necessary
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  Argument(s):
 - -          bnl - number of new lines to be printed          (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declaration(s) - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -  maxlin - maximum number of lines per page
 - -   nline - number of lines on current page
*/
  static short maxlin = 63;
  extern short nline;
/* - Add line print request to current page location and determine
     whether page eject is necessary */
  nline += nl;
  if(nline >= maxlin)
  { header();
    nline += nl;}
  return;
}
```

Function load_core_mtls

```
#include<stdio.h>
#include<string.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                            struct ascii_record *last;
                            ascii_string line;
                            struct ascii_record *next;
                           } a_record;

a_record *memory_ascii_record(int,int,a_record *);


a_record *load_core_mtls(char core_mtls[],int length_fin){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * l o a d _ c o r e _ m t l s  *  Loads Core Materials Dataset
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -  core_mtls - specification for file containing core mat-  (input)
 - -              materials definitions
 - - length_fin - length of file name                          (input)
 - -      first - pointer to first element in linked list     (output)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - -
 - -  Integer Variable(s)
 - -       c - single character used to process sucessive characters
 - -           from the input file
*/
  unsigned char c;
/* -  Character Variable(s)
 - -   record - string used to hold single lines from the input file
 - -        p - pointer to location in record
*/
  ascii_string record;
  char *p;
/* -  Structured Variable(s)
 - -        first - pointer to first member of linked list
 - -  ptr_current - pointer to current member of linked list
 - -     ptr_next - pointer to next member of linked list
*/
  a_record *first, *ptr_current, *ptr_next;
/* - FILE Variable(s)
      lu - pointer to core materials dataset
*/
  FILE *lu;
/* - Open Core Materials Dataset - - - - - - - - - - - - - - - - - - - - - */
  { short int i;
    for(i = 0;i < length_fin;i++){
      if(core_mtls[i] == ' '){
        core_mtls[i] = '\0';
        break;}
```

```
    }
  }
  lu = fopen(core_mtls,"r");
/* - Allocate First Structure for Processing File */
  ptr_current = memory_ascii_record(1,1,ptr_current);
  first = ptr_current;
  ptr_current->last = NULL;
  p = record;
  while((c = fgetc(lu)) != 10){
    *p = c;
    p++;}
  sprintf(p,"\n");
  strcpy(ptr_current->line,record);
  ptr_current->next = NULL;
/* - read balance of records in dataset */
  p = record;
  while((c = fgetc(lu)) != 255){
    *p = c;
    p++;
    if(c == 10){
      sprintf(p,"\n");
      ptr_next = memory_ascii_record(1,1,ptr_next);
      ptr_next->next = NULL;
      ptr_current->next = ptr_next;
      ptr_next->last = ptr_current;
      ptr_current = ptr_next;
      strcpy(ptr_current->line,record);
      p = record;}
  }
  return first;
}
```

Function load_fuel_material

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct s_material{
                          struct s_material *last;
                          int atomic_number;
                          int mass_number;
                          float weight_percentage;
                          char library_suffix[5];
                          struct s_material *next;
                         } ll_material;


ll_material *memory_s_material(int,int,ll_material *);
float *memory_float(int,int,float *);

ll_material *load_fuel_material(int *nft,char fprefix[],char dataset[]
 ,int nlat,int **lattice,float **fp_density){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * l o a d _ f u e l _ m a t e r i a l  *  Loads Contents of Fuel
 - -                                            Material Dataset into
```

```
 - -                                          Memory for Subsequent
 - -                                          Processing
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -        nft - Number of Fuel Rod Types in Lattice            (output)
 - -    fprefix - Prefix for Lattice Composition Database         (input)
 - -    dataset - Dataset Name for Fuel Intermediate Dataset of   (input)
 - -              Interest
 - -       nlat - lattice dimensionality                          (input)
 - -    lattice - Fuel Rod Type Map                              (output)
 - -          p - Vector of Pointers to Start of Linked List     (output)
 - -              Containing Material Inventories for each
 - -              Distinct Fuel Rod Type
 - - fp_density - densities for each fuel type                   (output)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  n - loop index
*/
  short int n;
/* - Character Variable(s)
 - -       c - single character for reading input file
 - -  buffer - string variable used to manage dataset processing
 - -      pb - pointer to buffer
 - -      fn - complete dataset file descriptor
 - -  header - header from dataset
*/
  unsigned char c;
  char *pb;
  ascii_string fn, buffer, header;
/* - File Variable(s)
 - -   ds - dataset stream pointer
 - - nout - output file
*/
  FILE *ds;
  extern FILE *nout;
/* - Structured Variable(s)
*/
  ll_material *p, *p_return;
/* - Put Dataset Name in C String Format - - - - - - - - - - - - - - - - - */
  { int len = 132, length;
    length = mchar(&len,dataset);
    dataset[length] = '\0';
  }
  strcpy(fn,fprefix);
  strcat(fn,dataset);
/* - Open File for Processing - - - - - - - - - - - - - - - - - - - - - - - */
  ds = fopen(fn,"r");
  if(!ds){
    lines(5);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout
     ," Load Fuel Material, Fuel Material Dataset not Found\n");
```

```
    fprintf(nout,"0Dataset Name:\n");
    fprintf(nout," %s\n",fn);
    abort();}
/* - Read Header Record */
  strcpy(buffer,"");
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while (c != 10);
  *pb = '\0';
  strcpy(header,buffer);
/* - Skip QA Record */
  do
    c = fgetc(ds);
  while (c != 10);
/* - Skip Fuel Rod Type Map Title */
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while (c != 10);
/* - Read Fuel Rod Type Map Line by Line */
  { short int j;
    int *plattice;
    ascii_string holder;
    plattice = *lattice;
    for( j = 1; j <= nlat; j++){
      pb = buffer;
      do{
        c = fgetc(ds);
        *pb = c;
        pb++;}
      while(c != 10);
      *pb = '\0';
      pb = buffer;
      for(n = 0; n < nlat; n++,plattice++){
        while(*pb == ' ') pb++;
        strcpy(holder,"");
        do{
          strncat(holder,pb,1);
          pb++;}
        while(*pb != ' ');
        pb++;
        sscanf(holder,"%i",plattice);}
    }
/* - Compute Number of Lattice Types */
    *nft = 0;
    plattice = *lattice;
    for( j = 1; j <= (nlat*nlat); j++,plattice++)
      if(*plattice > *nft) *nft = *plattice;
    p = memory_s_material(1,*nft,p);
```

```
  }
/* - Skip Title for Density Vector */
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while (c != 10);
/* - Allocate Memory for Density Vector */
  *fp_density = memory_float(1,*nft,*fp_density);
/* - Read Density Values */
  { short int nmax,i;
    ascii_string holder;
    float *p = *fp_density;
    n = 1;
    do{
      pb = buffer;
      do{
        c = fgetc(ds);
        *pb = c;
        pb++;}
      while(c != 10);
      *pb = '\0';
      pb = buffer;
      if((n+5) > *nft) nmax = *nft;
      else nmax = n+4;
      for( i = n; i <= nmax; i++,p++){
        while(*pb == ',') pb++;
        strcpy(holder,"");
        c = *pb;
        while((*pb != ',') && (c != 10)){
          strncat(holder,pb,1);
          pb++;
          c = *pb;}
        pb++;
        sscanf(holder,"%f",p);}
      n = nmax+1;
    }
    while(n <= *nft);
  }
/* - Skip Title for Fuel Material Compositions */
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while (c != 10);
/* - Read Each Materials for Each Fuel Rod Type */
/* - Allocate Memory for Vector of Linked Lists */
  p = memory_s_material(1,*nft,p);
  p_return = p;
  { unsigned char c;
    int index, n_entries, i, j;
    char label[6];
```

```
ascii_string holder;
ll_material *ptr_mtl, *ptr_last, *ptr_next;
for( n = 0; n < *nft; n++){
  ptr_mtl = p;
  ptr_last = NULL;
  ptr_mtl->atomic_number = 0;
  strcpy(ptr_mtl->library_suffix,"");
  ptr_mtl->mass_number = 0;
  ptr_mtl->weight_percentage = 0.0;
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while(c != 10);
  *pb = '\0';
  if(*nft > 1)
    sscanf(buffer,"%s %i %i",label,&index,&n_entries);
  else
    sscanf(buffer,"%i",&n_entries);
  for( i = 0; i < n_entries; i++){
    pb = buffer;
    do{
      c = fgetc(ds);
      *pb = c;
      pb++;}
    while(c != 10);
    *pb = '\0';
    pb = buffer;
    c = *pb;
    for( j = 1; j < 5; j++){
      strcpy(holder,"");
      while((*pb != ',') && (c != 10)){
        strncat(holder,pb,1);
        pb++;
        c = *pb;}
      switch (j) {
        case 1:
          { int value;
            sscanf(holder,"%i",&value);
            ptr_mtl->atomic_number = value;
            pb++;
          }
          break;
        case 2:
          { ascii_string value;
            sscanf(holder,"%s",value);
            strncpy(ptr_mtl->library_suffix,value,4);
            pb++;
          }
          break;
        case 3:
          { int value;
            sscanf(holder,"%i",&value);
```

```
                ptr_mtl->mass_number = value;
                pb++;
              }
              break;
            case 4:
              { float value;
                sscanf(holder,"%f",&value);
                ptr_mtl->weight_percentage = value;
                pb++;
              }
          }
        }
        ptr_mtl->next = NULL;
        ptr_mtl->last = ptr_last;
        ptr_last = ptr_mtl;
        if( i < (n_entries-1)){
          ptr_next = memory_s_material(1,1,ptr_next);
          ptr_mtl->next = ptr_next;
          ptr_mtl = ptr_next;
          ptr_mtl->atomic_number = 0;
          strcpy(ptr_mtl->library_suffix,"");
          ptr_mtl->mass_number = 0;
          ptr_mtl->weight_percentage = 0.0;
          }
      }
    p++;
    }
  }
  return p_return;
}
```

Function load_surface_usage_list

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;

surface_usage_list *memory_surface_usage_list(int,surface_usage_list *);

surface_usage_list *load_surface_usage_list(ascii_string label,int index
 ,ascii_string value,char mnemonic[4],ascii_string equivalent_label
 ,surface_usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - load_surface_usage_list - loads new values into surface usage list
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      label - character string to be added to new structure  (input)
 - -      index - index for character string                    (input)
 - -      value - definition of surface                         (input)
 - -   mnemonic - surface definition mnemonic                   (input)
 - - equivalent_label
 - -            - label of
 - -          p - pointer to memory locations                     (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Structured Variable(s)
 - -  p_new - pointer to new structure
*/
  surface_usage_list *p_new;
/* - Check for Initial Member of String - - - - - - - - - - - - - - - - - */
  if(p == NULL){
/* - Initial Member */
    p = memory_surface_usage_list(1,p);
    p->last = NULL;
    p->next = NULL;
    strcpy(p->label,label);
    strcpy(p->value,value);
    strcpy(p->mnemonic,mnemonic);
    strcpy(p->equivalent_label,equivalent_label);
    p->index = index;
    return p;}
/* - Addition to Linked List */
  else{
    p_new = memory_surface_usage_list(1,p_new);
    p->next = p_new;
    p_new->last = p;
    p = p_new;
    p->next = NULL;
    strcpy(p->label,label);
    strcpy(p->value,value);
    strcpy(p->mnemonic,mnemonic);
    strcpy(p->equivalent_label,equivalent_label);
    p->index = index;
    return p;}
}
```

Function load_usage_list

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
```

```
                    } usage_list;

usage_list *memory_usage_list(int,usage_list *);

usage_list *load_usage_list(char label[],int index,usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - load_usage_list - loads new values into usage list
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      label - character string to be added to new structure  (input)
 - -      index - index for character string                    (input)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Structured Variable(s)
 - -  p_new - pointer to new structure
*/
  usage_list *p_new;
/* - Check for Initial Member of String - - - - - - - - - - - - - - - - - - */
  if(p == NULL){
/* - Initial Member */
    p = memory_usage_list(1,p);
    p->last = NULL;
    p->next = NULL;
    strcpy(p->label,label);
    p->index = index;
    return p;}
/* - Addition to Linked List */
  else{
    p_new = memory_usage_list(1,p_new);
    p->next = p_new;
    p_new->last = p;
    p = p_new;
    p->next = NULL;
    strcpy(p->label,label);
    p->index = index;
    return p;}
}
```

Function mchar

```
      FUNCTION MCHAR(MAXLEN,BUFFER)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  M C H A R  * Determines the number of non-blank characters in
C - -                   character variable of total length MAXLEN
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -      MAXLEN - dimension of character variable              (input)
C - -      BUFFER - character variable                           (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type statement(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      CHARACTER*1 BLANK
      CHARACTER*(*) BUFFER
```

```
C - - Data statement(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      DATA BLANK/' '/
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      MCHAR = MAXLEN
      DO 10 NC = MAXLEN,1,-1
      IF(BUFFER(NC:NC) .NE. BLANK) THEN
        MCHAR = NC
        GO TO 20
      ENDIF
   10 CONTINUE
      MCHAR = 0
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - - - -
   20 CONTINUE
      END
```

Function memory_ascii_record

```
#include<stdio.h>
#include<malloc.h>
#include<errno.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                            struct ascii_record *last;
                            ascii_string line;
                            struct ascii_record *next;
                           } a_record;

a_record *memory_ascii_record(int operation,int number, a_record *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_ascii_record - manages memory requests for character storage
 - -                       for variables of the ascii_record structure
 - -                       type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures for which to allocate      (input)
 - -             memory
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
```

```
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = number*sizeof(a_record);
    p = (a_record *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_ascii_record: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_ascii_record: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += number*sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    return p;}
  else {
    size = number*sizeof(ascii_string);
    storage_c -= size;
    free(p);
    return p;}
}
```

Function memory_ascii_string

```
#include<stdio.h>
#include<string.h>
#include<malloc.h>
#include<errno.h>
typedef char ascii_string[133];

ascii_string *memory_ascii_string(int operation, int number
  ,ascii_string *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_ascii_string - manages memory requests for character storage
 - -                       for variables of the ascii_string type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of storage locations to request          (input)
 - -             (n.b., not used for return requests)
 - -         p - pointe to memory locations                       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
```

```
 - -          n - number of entries in proper size
 - -       size - number of bytes requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = ((size_t) number)*sizeof(ascii_string);
    p = (ascii_string *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_character: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_character: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += size;
    if(storage_c > storage_ct) storage_ct = storage_c;
    return p;}
  else {
    size = ((size_t) number)*sizeof(ascii_string);
    storage_c -= size;
    free(p);
    return p;}
}
```

Function memory_fg_list

```
#include <stdio.h>
#include <malloc.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct fuel_geometry{
                                ascii_string gds_name;
                                int latdim;
                                int nwr;
                                float cthick;
                                float asin;
                                float wgap;
```

```
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              char frcmat[6];
                              char fcmat[6];
                            } fg_list;

fg_list *memory_fg_list(int operation, int number, fg_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_fg_list - manages memory requests for storage for
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures to allocate                (input)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_r - current bytes of single-precision real storage
 - -              requested
 - -  storage_rt - maximum bytes of single-precision real storage
 - -              requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_r, storage_rt;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = number*sizeof(fg_list);
    p = (fg_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_fg_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
```

```
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_fg_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += 2*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    storage_r += 10*sizeof(float);
    if(storage_r > storage_rt) storage_rt = storage_r;
    storage_c += 12*sizeof(char)+sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    return p;}
  else {
    free(p);
    storage_i -= 2*sizeof(int);
    storage_r -= 10*sizeof(float);
    storage_c -= 12*sizeof(char)+sizeof(ascii_string);
    return p;}
}
```

Function memory_float

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>

float *memory_float(int operation, int number, float *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_integer - manages memory requests for single-precision real
 - -                  storage
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of storage locations to request          (input)
 - -             (n.b., not used for return requests)
 - -         p - pointer to memory location                       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   storage_r - current bytes of float storage requested
 - -  storage_rt - maximum bytes of float storage requested
*/
  extern int storage_r, storage_rt;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
```

```
    p = (float *)calloc((size_t)number,(size_t) sizeof(float));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_float: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_float: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_r += number;
    if(storage_r > storage_rt) storage_rt = storage_r;
    return p;}
  else {
    storage_r -= number;
    free(p);
    return p;};
}
```

Function memory_Integer

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>

int *memory_integer(int operation, int number, int *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_integer - manages memory requests for integer storage
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of storage locations to request          (input)
 - -             (n.b., not used for return requests)
 - -         p - pointer to memory location                       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
*/
  extern int storage_i, storage_it;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
```

```
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    p = (int *)calloc((size_t)number,(size_t) sizeof(int));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_integer: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_integer: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += number;
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    storage_i -= number;
    free(p);
    return p;};
}
```

Function memory_lattice_list

```
#include <stdio.h>
#include <malloc.h>
#include <errno.h>

typedef struct all{
                    struct all *last;
                    int basis_lattice_material_index;
                    int lattice_material_index;
                    struct all *next;
                   } augmented_lattice_list;

augmented_lattice_list *memory_lattice_list(int operation
 ,augmented_lattice_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_usage_list - manages memory requests for storage for vari-
 - -                     ables of the augmented_lattice_list structure
 - -                     type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
```

```
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -            n - number of entries in proper size
 - -         size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(augmented_lattice_list);
    p = (augmented_lattice_list *)calloc((size_t) 1
     ,(size_t) sizeof(augmented_lattice_list));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_lattice_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_lattice_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += 2*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    free(p);
    storage_i -= 2*sizeof(int);
    return p;}
}
```

Function memory_s_material

```
#include<stdio.h>
#include<malloc.h>
#include<errno.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
```

```
                        struct s_material *last;
                        int atomic_number;
                        int mass_number;
                        float weight_percentage;
                        char library_suffix[5];
                        struct s_material *next;
                       } ll_material;


ll_material *memory_s_material(int operation,int number,ll_material *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_s_material - manages memory requests for storage for vari-
 - -                     ables of the s_material structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures to process                 (input)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_r - current bytes of single-precision real storage
 - -              requested
 - -  storage_rt - maximum bytes of single-precision real storage
 - -              requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_r, storage_rt;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = number*sizeof(ll_material);
    p = (ll_material *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_s_material: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
```

```
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_s_material: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += 3*number*sizeof(char);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += 2*number*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    storage_r += number*sizeof(float);
    if(storage_r > storage_rt) storage_rt = storage_r;
    return p;}
  else {
    free(p);
    storage_c -= 3*number*sizeof(char);
    storage_i -= 2*number*sizeof(int);
    storage_r -= number*sizeof(float);
    return p;}
}
```

Function memory_surface_usage_list

```
#include <stdio.h>
#include <malloc.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;

surface_usage_list *memory_surface_usage_list(int operation
 , surface_usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_surface_usage_list - manages memory requests for storage for
 - -                             variables of the surface_usage_list
 - -                             structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
```

```
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -            n - number of entries in proper size
 - -         size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(surface_usage_list);
    p = (surface_usage_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += 3*sizeof(ascii_string)+4*sizeof(char);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    free(p);
    storage_c -= (3*sizeof(ascii_string)+4*sizeof(char));
    storage_i -= sizeof(int);
    return p;}
}
```

Function memory_usage_list

```
#include <stdio.h>
#include <malloc.h>
#include <errno.h>
```

```
typedef char ascii_string[133];

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

usage_list *memory_usage_list(int operation, usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_usage_list - manages memory requests for storage for vari-
 - -                     ables of the usage_list structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(usage_list);
    p = (usage_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_usage_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_usage_list: ");
        fprintf(nout,"Error Number %i Detected\n");
```

```
      }
      lines(2);
      abort();
    }
    storage_c += sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    free(p);
    storage_c -= sizeof(ascii_string);
    storage_i -= sizeof(int);
    return p;}
}
```

Function memsum

```
#include <stdio.h>

void memsum(){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  m e m s u m  * Edits the Dynamic Storage Usage
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  Integer Variable(s)
 - -   integer_storage - total integer storage expressed in bytes
 - -      real_storage - total real storage expressed in bytes
 - - character_storage - total character storage expressed in bytes
 - -     total_storage - total storage expressed in bytes
*/
  int integer_storage, real_storage, character_storage, total_storage;
  extern int storage_it, storage_rt, storage_ct;
 /* -  FILE Variable(s)
 - -  nout - pointer to code output file
*/
  extern FILE *nout;
/* - Compute Storage Totals in Bytes - - - - - - - - - - - - - - - - - - - - - */
  integer_storage = storage_it*sizeof(int);
  real_storage = storage_rt*sizeof(float);
  character_storage = storage_ct*sizeof(char);
  total_storage = integer_storage+real_storage+character_storage;
/* - Write the Total to Output - - - - - - - - - - - - - - - - - - - - - - - - */
  lines(8);
  fprintf(nout,"0Total Dynamic Memory Usage in Bytes:\n");
  fprintf(nout,"\n");
  fprintf(nout,"   Integer = %7i \n",integer_storage);
  fprintf(nout,"      Real = %7i \n",real_storage);
  fprintf(nout," Character = %7i \n",character_storage);
  fprintf(nout,"             ------- \n");
  fprintf(nout,"     Total = %7i \n",total_storage);
  return;
}
```

Function rollup_llm

```
#include<stdlib.h>

typedef struct s_material{
                          struct s_material *last;
                          int atomic_number;
                          int mass_number;
                          float weight_percentage;
                          char library_suffix[5];
                          struct s_material *next;
                         } ll_material;

ll_material *memory_s_material(int,int,ll_material *);

void rollup_llm(ll_material *p){
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - rollup_llm - returns memory associated with a linked list with a
 - -              s_material structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -         p - pointer to start of linked list                  (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Structured Variable(s)
 - -  last - pointer to previous structure in linked list
*/
   ll_material *last;
/* - Search for End of Linked List - - - - - - - - - - - - - - - - - - - - - */
   while(p->next != NULL) p = p->next;
/* - Sweep through List Backwards, Returning Memory - - - - - - - - - - - - */
  while(p->last != NULL){
    last = p->last;
    p = memory_s_material(-1,1,p);
    p = last;
  }
  return;
}
```

Function search_surface_usage_list

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;

void search_surface_usage_list(int option,ascii_string label,int *index
 ,ascii_string value,char mnemonic[4],ascii_string equivalent_label
 ,surface_usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - search_surface_usage_list - searches a surface usage linked list
 - -                             seeking match
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     option - flag to indicate processing to be performed    (input)
 - -              (0 - search for label based on index,
 - -               1 - search for index based on label,
 - -               2 - serach for index based on surface defini- (input)
 - -                   tion)
 - -      label - character string either to be sought or          (i|o)
```

```
 - -                returned
 - -       index - index either to be sought or returned          (i|o)
 - -       value - definition of surface                        (input)
 - -   mnemonic - mnemonic for surface                          (input)
 - - equivalent_label
 - -             - label for equivalent surface                 (input)
 - -           p - pointer to start of linked list              (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - FILE Variable(s)
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Search based on Index - - - - - - - - - - - - - - - - - - - - - - - - */
  if(option == 0){
    do{
      if(p->index == *index){
        strcpy(label,p->label);
        return;}
      p = p->next;
    } while(p->next != NULL);
/* - Process the Last Entry */
    if(p->index == *index){
      strcpy(label,p->label);
      return;}
    else{
      lines(2);
      fprintf(nout
       ,"0***  F A T A L  E R R O R  ***  Function search_usage_list:");
          fprintf(nout," No match found for index = %i\n",*index);
      abort();}
  }
  if(option == 1){
    do{
      if(strcmp(label,p->label) == 0){
        *index = p->index;
        return;}
      p = p->next;
    } while(p->next != NULL);
/* - Process the Last Entry */
    if(strcmp(label,p->label) == 0){
      *index = p->index;
      return;}
    else{
      *index = 0;
    }
  }
  if(option == 2){
    do{
      if((strcmp(value,p->value) == 0) &&
         (strcmp(mnemonic,p->mnemonic) == 0)){
        *index = p->index;
        strcpy(equivalent_label,p->label);
```

```
        return;}
      p = p->next;
    } while(p->next != NULL);
/* - Process the Last Entry */
    if((strcmp(value,p->value) == 0) &&
       (strcmp(mnemonic,p->mnemonic) == 0)){
      *index = p->index;
      strcpy(equivalent_label,p->label);
      return;}
    else
      *index = 0;
  }
}
```

Function search_usage_list

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

void search_usage_list(int option,char label[],int *index
,usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - search_usage_list - searches a usage linked list seeking match
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     option - flag to indicate processing to be performed    (input)
 - -              (0 - search for label based on index,
 - -               1 - search for index based on label)
 - -      label - character string either to be sought or          (i|o)
 - -              returned
 - -      index - index either to be sought or returned            (i|o)
 - -          p - pointer to start of linked list                (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - FILE Variable(s)
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Search based on Index - - - - - - - - - - - - - - - - - - - - - - - - - */
  if(option == 0){
    do{
      if(p->index == *index){
        strcpy(label,p->label);
        return;
      }
```

```
      p = p->next;
    } while(p->next != NULL);
/* - Process the Last Entry */
    if(p->index == *index){
      strcpy(label,p->label);
      return;}
    else{
      lines(2);
      fprintf(nout
       ,"0***  F A T A L  E R R O R  *** -- Function search_usage_list:");
          fprintf(nout," No match found for index = %i\n",*index);
      abort();}
  }
  else{
    do{
      if(strcmp(label,p->label) == 0){
        *index = p->index;
        return;
      }
      p = p->next;
    } while(p->next != NULL);
/* - Process the Last Entry */
    if(strcmp(label,p->label) == 0){
      *index = p->index;
      return;}
    else{
      *index = 0;
    }
  }
}
```

2.3. Input Routines

Subroutine bpchek

```
      SUBROUTINE BPCHEK(NROWB,NCOLB,NROWBP,NCOLBP,VALIDB,BLADEP)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  B P C H E K  *  Determines whether Blade Positions are Valid
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        NROWB - Number of Control Blade Rows in Full Core  (input)
C - -        NCOLB - Number of Control Blade Columns in Full    (input)
C - -                Core
C - -       NROWBP - Number of Control Blade Rows in Problem    (input)
C - -       NCOLBP - Number of Control Blade Columns in Problem (input)
C - -       VALIDB - Integer Array Containing Valid Locations   (input)
C - -                for Control Blades
C - -       BLADEP - Integer Array Containing Blade Insertion   (input)
C - -                 Values
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   ERROR - Logical Flag Indicating that Fatal Error has been
C - -           Detected
C - - MESSAGE - String for Messages
      LOGICAL ERROR
      INTEGER VALIDB(NCOLB,NROWB), BLADEP(NCOLBP,NROWBP)
      CHARACTER*132 MESSAGE
C - - Sweep through Map - - - - - - - - - - - - - - - - - - - - - - - - -
      ERROR = .FALSE.
      DO J = 1,NROWBP
        DO I = 1,NCOLBP
          IF(VALIDB(I,J) .EQ. 0) THEN
            IF(BLADEP(I,J) .EQ. -1) THEN
              CONTINUE
            ELSE
              ERROR = .TRUE.
              CALL lines(3)
              MESSAGE =
     2 '0***  F A T A L  E R R O R  ***  SUBR. BPCHEK -- Control'
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Blade Assignment to Invalid Location (')')
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') I
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'('',')')
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') I
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'('')'')')
              CALL fortran_message(MESSAGE)
              MESSAGE = ' Blade Position = '
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'(I2)') BLADEP(I,J)
              CALL fortran_message(MESSAGE)
            ENDIF
          ENDIF
        ENDDO
```

```
      ENDDO
C - - Check for Valid Range
      DO J = 1,NROWBP
        DO I = 1,NCOLBP
          IF(VALIDB(I,J) .NE. 0) THEN
            IF((BLADEP(I,J) .LT. 0) .OR. (BLADEP(I,J) .GT. 48)) THEN
              ERROR = .TRUE.
              CALL lines(2)
              MESSAGE =
     2 '0***  F A T A L  E R R O R  ***  SUBR. BPCHEK -- '
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Invalid Blade Position: Location (')')
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') I
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'('',')')
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') I
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'('')'')')
              CALL fortran_message(MESSAGE)
            ENDIF
          ENDIF
        ENDDO
      ENDDO
C - - Abort for Assignment Errors - - - - - - - - - - - - - - - - - - - -
      IF(ERROR) THEN
        CALL lines(2)
        CALL abort()
      ENDIF
C - - End of Processing - - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Function cbundle

```
      FUNCTION CBUNDLE(NCOL,NROW,MAX,MAP,LIST)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * C B U N D L E *  Determines the Number of Unique Fuel As-
C - -                      sembly Types in the Core
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -         NCOL - Number of "Columns" in Core Map            (input)
C - -         NROW - Number of "Rows" in Core Map               (input)
C - -          MAX - Maximum Possible Number of Unique Assem-   (input)
C - -                blies
C - -          MAP - Map of Indices for Core Locations          (input)
C - -                (integer)
C - -         LIST - Scratch Array for Accmulating a List of  (scratch)
C - -                Unique Fuel Assembly Indices
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MATCH - Logical Variable used to Indicate that a Fuel Assembly
C - -          Identifer Already exists in the Unique Index List
      LOGICAL MATCH
```

```
      INTEGER CBUNDLE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION LIST(MAX)
      DIMENSION MAP(NCOL,NROW)
C - - Sweep through Map Looking for Unique Fuel Assembly Identifiers -
      NU = 0
      DO J = 1,NROW
        DO I = 1,NCOL
          IF(MAP(I,J) .NE. 0) THEN
            IF(NU .EQ. 0) THEN
              NU = 1
              LIST(NU) = MAP(I,J)
            ELSE
              MATCH = .FALSE.
              DO N = 1,NU
                IF(LIST(N) .EQ. MAP(I,J)) MATCH = .TRUE.
              ENDDO
              IF(.NOT. MATCH) THEN
                NU = NU+1
                LIST(NU) = MAP(I,J)
              ENDIF
            ENDIF
          ENDIF
        ENDDO
      ENDDO
      CBUNDLE = NU
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Function clattice

```
      FUNCTION CLATTICE(NAXP1,NBUNDLE,MAX,LVECT,LIST)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * C L A T T I C E *  Determines the Number of Unique Fuel
C - -                        Lattices in the Core
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        NAXP1 - Number of Axial Nodes in the Core Plus One (input)
C - -      NBUNDLE - Number of Unique Fuel Assemblies in the    (input)
C - -                Core
C - -          MAX - Maximum Possible Number of Unique Lat-     (input)
C - -                tices
C - -        LVECT - Vectors Assigning Lattice Indices to Fuel  (input)
C - -                Assemblies
C - -         LIST - Scratch Array for Accmulating a List of  (scratch)
C - -                Unique Fuel Lattice Indices
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MATCH - Logical Variable used to Indicate that a Fuel Assembly
C - -          Identifer Already exists in the Unique Index List
      LOGICAL MATCH
      INTEGER CLATTICE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
```

```
      DIMENSION LIST(MAX)
      DIMENSION LVECT(NAXP1,NBUNDLE)
C - - Sweep through Fuel Assembly Loading Vectors Looking for Unique
C - - Lattice Identifiers
      NU = 0
      DO J = 1,NBUNDLE
C - - Note that the first entry for each Fuel Assembly is the Fuel
C - - Assembly Sequential Index
        DO I = 2,NAXP1
          IF(LVECT(I,J) .NE. 0) THEN
            IF(NU .EQ. 0) THEN
              NU = 1
              LIST(NU) = LVECT(I,J)
            ELSE
              MATCH = .FALSE.
              DO N = 1,NU
                IF(LIST(N) .EQ. LVECT(I,J)) MATCH = .TRUE.
              ENDDO
              IF(.NOT. MATCH) THEN
                NU = NU+1
                LIST(NU) = LVECT(I,J)
              ENDIF
            ENDIF
          ENDIF
        ENDDO
      ENDDO
      CLATTICE = NU
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Function echo

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];

void echo(char inname[8], char fin[133]){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -   *  e c h o  *  Copies the input file to the output stream
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -   Argument(s):
 - -       inname - Name for Input FORTRAN NAMELIST              (input)
 - -          fin - Name for Input File                          (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declaration(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -  infind - Flag to Indicate that the FORTRAN NAMELIST Directive
 - -           has been Encountered
*/
  unsigned short infind = 0;
/* - Character Variable(s)
 - -          c - single character used to read from input file
 - -          p - pointer to character variable
```

```
 - -     buffer - string variable used to copy files from input file
 - - case_title - title for analysis
*/
  unsigned char c;
  char *p;
  char buffer[133] = "";
  extern codenm[8];
  extern ascii_string case_title;
/* - FILE Variable(s)
 - -  nin - input file
*/
  FILE *nin;
  extern FILE *nout;
/* - - Echo input stream - - - - - - - - - - - - - - - - - - - - - - - - - */
  nin = fopen(fin,"r");
  lines(4);
  fprintf(nout,"0Input to %s :\n",codenm);
  fprintf(nout,"\n");
/* - Read Title Line and Copy to Output File*/
  p = buffer;
  do
    {. c = fgetc(nin);
      *p = c;
      p++;}
  while(c != 10);
  fprintf(nout," ");
  p = buffer;
  c = *p;
  while(c != 10){
    fputc(*p,nout);
    p++;
    c = *p;
  }
  fprintf(nout,"\n");
  strncpy(case_title,buffer,strlen(buffer));
  lines(1);
/* - Read Second Line and Look for FORTRAN NAMELIST Input Directive */
  do
  { strcpy(buffer,"");
    p = buffer;
    do
    { c = fgetc(nin);
      *p = c;
      p++;}
    while((c != 10) && (c != 255));
    if(strstr(buffer,inname) != NULL) infind = 1;
    if(c != 255) fprintf(nout," ");
    if(c != 255){
      p = buffer;
      c = *p;
      while(c != 10){
        fputc(*p,nout);
        p++;
        c = *p;
```

```
      }
      fprintf(nout,"\n");
      lines(1);
    }
  }
  while(c != 255);
  if(!infind)
  {  lines(4);
     fprintf(nout,"0*** F A T A L  E R R O R  ***  Function echo --");
     fprintf(nout," No Namelist Directive Encountered.\n");
     abort();
  }
  rewind(nin);
  fclose(nin);
  return;
}
```

Subroutine ldlv

```
      SUBROUTINE LDLV(NBUNDLE,NAXP1,LU,LVECT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  L D L V *  Reads Vectors giving Lattice Assignments to Fuel
C - -                 Assemblies
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -      NBUNDLE - Number of Fuel Assemblies in Core          (input)
C - -        NAXP1 - Number of Axial Nodes in Core +1           (input)
C - -           LU - Logical Unit from which to Read Vectors    (input)
C - -        LVECT - Lattice Loading Vectors for each Fuel     (output)
C - -                Assembly
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MESSAGE - Character String for Messages
      CHARACTER*132 MESSAGE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION LVECT(NAXP1,NBUNDLE)
C - - Skip Header Record - - - - - - - - - - - - - - - - - - - - - - - -
      READ(LU,'(1X)')
      DO NB=1,NBUNDLE
        READ(LU,*,END=500) (LVECT(I,NB),I=1,NAXP1)
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 MESSAGE = '0***  F A T A L  E R R O R  ***  SUBR. LDLV --'
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Premature End-of-File Encountered on Logical Unit'')')
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' '',I2)') LU
      CALL lines(5)
      CALL fortran_message(MESSAGE)
      CALL abort()
      END
```

Subroutine ldlvr

```
      SUBROUTINE LDLVR(NBUNDLE,NAXP1,LU,LVECT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * L D L V *  Reads Vectors giving Lattice Assignments to Fuel
C - -                Assemblies (Real Version of LDLV)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -      NBUNDLE - Number of Fuel Assemblies in Core          (input)
C - -        NAXP1 - Number of Axial Nodes in Core +1           (input)
C - -           LU - Logical Unit from which to Read Vectors    (input)
C - -        LVECT - Lattice Loading Vectors for each Fuel     (output)
C - -                Assembly
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MESSAGE - Character String for Messages
      CHARACTER*132 MESSAGE
      REAL LVECT(NAXP1,NBUNDLE)
C - - Skip Header Record - - - - - - - - - - - - - - - - - - - - - - - -
      READ(LU,'(1X)')
      DO NB=1,NBUNDLE
        READ(LU,*,END=500) (LVECT(I,NB),I=1,NAXP1)
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 MESSAGE = '0***  F A T A L  E R R O R  ***  SUBR. LDLVR --'
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Premature End-of-File Encountered on Logical Unit'')')
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' '',I2)') LU
      CALL lines(5)
      CALL fortran_message(MESSAGE)
      CALL abort()
      END
```

Subroutine lodct

```
      SUBROUTINE LODCT(NCOLP,NROWP,NAXP1,NLATTICG,NLATTICM,GMAP,MMAP
     2 ,LGVECT,LMVECT,CTABLE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * L O D C T *  Load Correspondence Table Matching Lattice
C - -                  Geometry Types with Lattice Material Types
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        NCOLP - Number of Columns in Core Fuel Assembly    (input)
C - -                Map
C - -        NROWP - Number of Rows in Core Fuel Assembly Map   (input)
C - -        NAXP1 - Number of Axial Nodes + 1                  (input)
C - -     NLATTICG - Number of Unique Lattice Geometry Types    (input)
C - -     NLATTICM - Number of Unique Lattice Material Types    (input)
C - -         GMAP - Fuel Assembly Geometry Index Map           (input)
C - -         MMAP - Fuel Assembly Material Index Map           (input)
C - -       LGVECT - Vectors Provides Lattice Geometry Assign-  (input)
```

```
C - -                 ment Indices for a Given Fuel Assembly
C - -                 Geometry Index
C - -        LMVECT - Vectors Provides Lattice Material Assign-  (input)
C - -                 ment Indices for a Given Fuel Assembly
C - -                 Material Index
C - -        CTABLE - Correspondence Table Providing Lattice    (output)
C - -                 Geometry Index for a Given Lattice Mater-
C - -                 ial Index
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MATCH - Flag to Indicate that a Lattice Material/Lattice
C - -          Geometry Pair has Already been Added to CTABLE
      LOGICAL MATCH
      INTEGER GMAP(NCOLP,NROWP),CTABLE(2,NLATTICM)
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MMAP(NCOLP,NROWP),LGVECT(NAXP1,NLATTICG)
     2 ,LMVECT(NAXP1,NLATTICM)
C - - Sweep Over Entire Core - - - - - - - - - - - - - - - - - - - - - - -
      CALL INVALI((2*NLATTICM),CTABLE,0)
      IC = 0
      DO J = 1,NROWP
        DO I = 1,NCOLP
          IF(MMAP(I,J) .NE. 0) THEN
C - - Examine Each Node of Fuel Assembly Loading Vector
            DO K = 2,NAXP1
              LGV = LGVECT(K,GMAP(I,J))
              LMV = LMVECT(K,MMAP(I,J))
              IF(IC .EQ. 0) THEN
                IC = IC+1
                CTABLE(1,IC) = LMV
                CTABLE(2,IC) = LGV
              ELSE
                MATCH = .FALSE.
                DO N = 1,IC
                  IF((CTABLE(1,N) .EQ. LMV) .AND.
     2                (CTABLE(2,N) .EQ. LGV)) MATCH = .TRUE.
                ENDDO
                IF(.NOT. MATCH) THEN
                  IC = IC+1
                  CTABLE(1,IC) = LMV
                  CTABLE(2,IC) = LGV
                ENDIF
              ENDIF
            ENDDO
          ENDIF
        ENDDO
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine mapchek

```
      SUBROUTINE MAPCHEK(NROW,NCOL,NROWP,NCOLP,VALID,MAP)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * M A P C H E K *  Determines whether Input Fuel Assembly
C - -                      Locations are Valid and if all Valid
C - -                      Locations have Fuel Assemblies Indices
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -         NROW - Number of Rows in Full Core                (input)
C - -         NCOL - Number of Columns in Full Core             (input)
C - -        NROWP - Number of Rows in Problem                  (input)
C - -        NCOLP - Number of Columns in Problem               (input)
C - -          MAP - Integer Array Containing Fuel Assembly     (input)
C - -                Identifier Map
C - -        VALID - Integer Array Containing Valid Locations   (input)
C - -                in Core for Fuel Assemblies
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -    ERROR - Logical Flag Indicating that Fatal Error has been
C - -            Detected
C - -  MESSAGE - String for Message
      LOGICAL ERROR
      INTEGER VALID(NCOL,NROW)
      CHARACTER*132 MESSAGE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MAP(NCOLP,NROWP)
C - - Sweep through Map - - - - - - - - - - - - - - - - - - - - - - - - -
      ERROR = .FALSE.
      DO J = 1,NROWP
        DO I = 1,NCOLP
          IF(VALID(I,J) .EQ. 0) THEN
            IF(MAP(I,J) .EQ. 0) THEN
              CONTINUE
            ELSE
              ERROR = .TRUE.
              CALL lines(3)
              MESSAGE =
     2 '0***  F A T A L  E R R O R  ***  SUBR. MAPCHEK -- Fuel '
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'(''Assembly Assignment to Invalid Location ('')')
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') I
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'('','')')
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') J
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'('')'')')
              CALL fortran_message(MESSAGE)
              MESSAGE = 'Fuel Assmebly Index = '
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+2)),'(I3)') MAP(I,J)
              CALL fortran_message(MESSAGE)
            ENDIF
```

```
          ELSE
            IF(MAP(I,J) .NE. 0) THEN
              CONTINUE
            ELSE
              ERROR = .TRUE.
              CALL lines(3)
              MESSAGE =
     2 '0***  F A T A L  E R R O R  ***  SUBR. MAPCHEK -- No '
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'(''Fuel Assembly Assignment to Valid Location (')')
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') I
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'(','')')
              LOC = MCHAR(132,MESSAGE)+1
              WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') J
              WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):),'(')'')')
              CALL fortran_message(MESSAGE)
            ENDIF
          ENDIF
        ENDDO
      ENDDO
C - - Abort for Assignment Errors - - - - - - - - - - - - - - - - - - - - -
      IF(ERROR) THEN
        CALL lines(2)
        CALL abort()
      ENDIF
C - - End of Processing - - - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine r2dmap

```
      SUBROUTINE R2DMAP(NCOL,NROW,MAP,LU)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  R 2 D M A P *  Reads Two-dimensional Map giving Fuel
C - -                     Assembly Locations
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -          NCOL - Number of "Columns" in Core Map            (input)
C - -          NROW - Number of "Rows" in Core Map               (input)
C - -           MAP - Map of Indices for Core Locations         (output)
C - -                 (integer)
C - -            LU - Logical Unit from which to Read Map        (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MESSAGE - message to pass to output file
      INTEGER MAP(NCOL,NROW)
      CHARACTER*132 MESSAGE
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DO J = 1,NROW
        READ(LU,*,END=500) (MAP(I,J),I=1,NCOL)
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
```

```
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - - -
  500 CALL lines(3)
      MESSAGE = '0***  F A T A L  E R R O R  ***  SUBR. R2DMAP -- '
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'(''Premature End-of-File Encountered on Logical Unit '')')
      LOC = MCHAR(132,MESSAGE)
      WRITE(MESSAGE(LOC:(LOC+1)),'(I2)') LU
      CALL fortran_message(MESSAGE)
      CALL abort()
      END
```

Subroutine rblade

```
      SUBROUTINE RBLADE(LENFN,BLADE_DB,LUGEOM,NTUBE,CBSPAN,ATID,ATOD
     2 ,CBTHICK,TRSPAN,TRTHICK,WSTHICK,CBLENGTH,NCS,CSOFF,CSWIDTH
     3 ,CBPMAT,ATMAT,CBSMAT,CBTRMAT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * R B L A D E *  Reads Dataset Containing Control Blade
C - -                    Geometry Dataset
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        LENFN - Length of File Name                        (input)
C - -     BLADE_DB - Name of File Containing Appropriate Core-  (input)
C - -                layout Database
C - -       LUGEOM - Logical Unit Number for Blade Geometry     (input)
C - -                Dataset
C - -        NTUBE - Number of Absorber Tubes in All Four      (output)
C - -                Wings of Blade
C - -       CBSPAN - Control Blade Span (Tie Rod Center to     (output)
C - -                Tip)
C - -         ATID - Absorber Tube Inner Diameter (cm)         (output)
C - -         ATOD - Absorber Tube Outer Diameter (cm)         (output)
C - -      CBTHICK - Blade Wing Thickness (cm)                 (output)
C - -       TRSPAN - Tie Rod Span (cm)                         (output)
C - -      TRTHICK - Tie Rod Thickness (cm)                    (output)
C - -      WSTHICK - Wing Thickness (cm)                       (output)
C - -     CBLENGTH - Active Absrober Length (cm)               (output)
C - -          NCS - Number of Central Stiffeners per Wing     (output)
C - -        CSOFF - Central Stiffener Offset                  (output)
C - -      CSWIDTH - Central Stiffener Width                   (output)
C - -       CBPMAT - Identifier for Blade Poison Material      (output)
C - -        ATMAT - Identifier for Blade Absorber Tube Mat-   (output)
C - -                erial
C - -       CBSMAT - Identifier for Blade Sheath Material      (output)
C - -      CBTRMAT - Identifier for Blade Tie Rod Material     (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      CHARACTER*6 CBPMAT,ATMAT,CBSMAT,CBTRMAT
      CHARACTER*(LENFN) BLADE_DB
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /BLADE/ NTUBE,CBSPAN,ATID,ATOD,CBTHICK,TRSPAN,TRTHICK
     2 ,CBLENGTH,WSTHICK,NCS,CSOFF,CSWIDTH,CBPMAT,ATMAT,CBSMAT,CBTRMAT
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
      CALL FTOPEN(LUGEOM,BLADE_DB)
C - - Skip Record Containing Dataset Title
      READ(LUGEOM,'(1X)')
C - - Read NAMELIST Input Block
      READ(LUGEOM,BLADE)
C - - Close Dataset
      CALL FTCLOSE(LUGEOM)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

Subroutine readin

```
      SUBROUTINE READIN(FIN,LENFN,CORE_DB,NIN,LUCGEOM,NAXIAL,AFL,NROW
     2 ,NCOL,NROWP,NCOLP,APITCH,SOD,STHICK,VOD,VTHICK,TUTPR,TCGR,BLTPR
     3 ,BCPR,LPREFIX,FPREFIX,CORE_MTLS,MVESSEL,MSHROUD,MTG,MCP,MUTP
     4 ,MLTP,CORE_F,BLADE_DB,MIGT,DTOD,DTID,RHO,RHOBYP,TEMPK)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  R E A D I N  *  Reads NAMELIST input to program -- including
C - -                     the core geometry file (defined as function
C - -                     for compatibility with C invocation)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -          FIN - File Name for Input File                    (input)
C - -        LENFN - Length of File Name                         (input)
C - -      CORE_DB - Name of File Containing Appropriate Core-  (output)
C - -                layout Database
C - -          NIN - Logical Unit Number for Input File          (input)
C - -      LUCGEOM - Logical Unit Number for Core Geometry       (input)
C - -                Dataset File
C - -       NAXIAL - Number of Axial Nodes in Active Fuel       (output)
C - -          AFL - Active Fuel Length (cm)                    (output)
C - -         NROW - Number of Rows in Full Core                (output)
C - -         NCOL - Number of Columns in Full Core             (output)
C - -        NROWP - Number of Rows in Core Represented in the (output)
C - -                Problem
C - -        NCOLP - Number of Columnes in the Core Repre-      (output)
C - -                sented in the Problem
C - -       APITCH - Assembly Pitch (cm)                        (output)
C - -          SOD - Outer Radius of Core Shroud (cm)           (output)
C - -       STHICK - Core Shroud Thickness (cm)                 (output)
C - -          VOD - Pressure Vessel Outer Radius (cm)          (output)
C - -       VTHICK - Pressure Vessel Thickness (cm)             (output)
C - -        TUTPR - Top of the Upper Tie Plate Region (cm)     (output)
C - -         TCGR - Top of the Core Grid Region (cm)           (output)
C - -        BLTPR - Bottom of the Lower Tie Plate Region (cm)  (output)
C - -         BCPR - Bottom of the Core Plate Region (cm)       (output)
C - -      LPREFIX - File Description Prefix for Lattice Geo-   (output)
C - -                metry Files
C - -      FPREFIX - File Description Prefix for Fuel Materi    (output)
C - -                al Intermediate Datasets
C - -    CORE_MTLS - File Specification for Core Materials      (output)
C - -                Dataset
C - -      MVESSEL - Material Identifier for Vessel             (output)
C - -      MSHROUD - Material Identifier for Core Shroud        (output)
```

```
C - -          MTG - Material Identifier for Core Top Guide     (output)
C - -                Region
C - -          MCP - Material Identifier for Core Plate         (output)
C - -                Region
C - -         MUTP - Material Identifier for Upper Tie Plate    (output)
C - -                Region
C - -         MLTP - Material Identifier for Lower Tie Plate    (output)
C - -                Region
C - -       CORE_F - Fraction of Core in Problem                (output)
C - -                (1 - full, 2 - half, 4 - quarter)
C - -     BLADE_DB - Dataset for Control Blade used in the      (output)
C - -                Problem
C - -         MIGT - Material Identifier for Incore Guide Tube (output)
C - -         DTOD - Incore Guide Tube Outer Diameter (cm)      (output)
C - -         DTID - Incore Guide Tube Inner Diameter (cm)      (output)
C - -          RHO - In-channel Water Density (g/cc)            (output)
C - -       RHOBYP - Bypass Water Density (g/cc)                (output)
C - -        TEMPK - Problem Temperature (Kelvin)               (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -     NIN - Logical Unit Number for Input File
C - - LUCGEOM - Logical Unit Number for Core-geometry Dataset File
      INTEGER NIN,CORE_F
      CHARACTER*6 MVESSEL,MSHROUD,MTG,MCP,MUTP,MLTP,MIGT
      CHARACTER*132 CORE_DB,LPREFIX,FPREFIX,CORE_MTLS,BLADE_DB
      CHARACTER*(LENFN) FIN
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /LINKIN/ CORE_DB,NAXIAL,AFL,NROWP,NCOLP,LPREFIX,FPREFIX
     2 ,CORE_MTLS,CORE_F,BLADE_DB,RHO,RHOBYP,TEMPK,MUTP,MLTP
      NAMELIST /CORE/ NCOL,NROW,APITCH,SOD,STHICK,VOD,VTHICK,TUTPR
     2 ,TCGR,BLTPR,BCPR,MVESSEL,MSHROUD,DTOD,DTID,MTG,MCP,MIGT
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FTOPEN(NIN,FIN)
C - - Skip Record Containing Case Title
      READ(NIN,'(1X)')
C - - Read NAMELIST Input Block
      READ(NIN,LINKIN)
C - - Process Core-geometry File Namelist Input Section - - - - - - - -
      CALL FTOPEN(LUCGEOM,CORE_DB)
C - - Skip Header Record
      READ(LUCGEOM,'(1X)')
C - - Read NAMELIST Input
      READ(LUCGEOM,CORE)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

Subroutine rlattice

```
      SUBROUTINE RLATTICE(LENFN,FUEL_DB,LUGEOM,CTHICK,ASIN,WGAP,NGAP
     2 ,CRADIUS,FSRD,CFSRD,RPITCH,COD,CLD,POD,FRCMAT,FCMAT,LATDIM,NWR)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * R L A T T I C E *  Reads Dataset Containing Lattice
C - -                        Geometry Dataset
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        LENFN - Length of File Name                          (input)
C - -      FUEL_DB - Name of File Containing Appropriate          (input)
C - -                Lattice-layout Database
C - -       CTHICK - Channel Thickness (cm)                      (output)
C - -         ASIN - Inner Span of Channel (cm)                  (output)
C - -         WGAP - Wide Gap Thickness (cm)                     (output)
C - -         NGAP - Narrow Gap Thickness (cm)                   (output)
C - -      CRADIUS - Inner Radius of Channel Corner (cm)         (output)
C - -         FSRD - Clad Surface to Clad Surface Separation     (output)
C - -                (cm)
C - -        CFSRD - Clad Surface to Inner Channel Surface       (output)
C - -                Separation (cm)
C - -       RPITCH - Fuel Rod Pitch (cm)                         (output)
C - -          COD - Cladding Outer Diameter (cm)                (output)
C - -          CLD - Cladding Thickness (cm)                     (output)
C - -          POD - Fuel Pellet Outer Diameter (cm)             (output)
C - -       FRCMAT - Material Identifier for Fuel Rod Cladding   (output)
C - -        FCMAT - Material Identifier for Channel             (output)
C - -       LATDIM - Lattice Dimensionality                      (output)
C - -          NWR - Number of Water Rods                        (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      REAL NGAP
      CHARACTER*6 FRCMAT,FCMAT
      CHARACTER*(LENFN) FUEL_DB
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /FUEL/ CTHICK,ASIN,WGAP,NGAP,CRADIUS,FSRD,CFSRD,RPITCH
     2 ,COD,CLD,POD,LATDIM,NWR,FRCMAT,FCMAT
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FTOPEN(LUGEOM,FUEL_DB)
C - - Skip Record Containing Dataset Title
      READ(LUGEOM,'(1X)')
C - - Read NAMELIST Input Block
      READ(LUGEOM,FUEL)
C - - Close Dataset
      CALL FTCLOSE(LUGEOM)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

Subroutine strngr

```
      SUBROUTINE STRNGR(LU,LEN,NSTR,VECTOR)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * S T R N G R *  Reads Character Strings from FORTRAN Logical
C - -                    Unit
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -            LU - Logical Unit from which to Read Character  (input)
C - -                 Strings
C - -           LEN - Length of Character Strings                (input)
C - -          NSTR - Number of FORTRAN Records to Read          (input)
C - -        VECTOR - Character String Storage Locations        (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MESSAGE - Character String for Message
      CHARACTER*132 MESSAGE
      CHARACTER*(LEN) VECTOR(NSTR)
C - - Skip Header Record - - - - - - - - - - - - - - - - - - - - - - - -
      READ(LU,'(1X)')
      DO N=1,NSTR
        READ(LU,'(A)',END=500) VECTOR(N)
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 MESSAGE = '0***  F A T A L  E R R O R  ***  SUBR. STRNGR --'
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Premature End-of-File Encountered on Logical Unit'')')
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' '',I2)') LU
      CALL lines(5)
      CALL fortran_message(MESSAGE)
      CALL abort()
      RETURN
      END
```

2.4. Input Editing Routines

Function bladedb_edt

```
#include <stdio.h>
#include <string.h>

void bladedb_edt(int ntube,float cbspan,float atid,float atod
 ,float cbthick,float trspan,float trthick,float wsthick
 ,float cblength,int ncs,float csoff,float cswidth,char cbpmat[]
 ,char atmat[],char cbsmat[],char cbtrmat[]){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * b l a d e d b _ e d t  *  Edits Contents of Control Blade
 - -                              Geometry Dataset
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      ntube - number of absorber tubes in control blade    (input)
 - -     cbspan - control blade span                           (input)
 - -       atid - blade absorber tube inner diameter (cm)      (input)
 - -       atod - blade absorber tube outer diameter (cm)      (input)
 - -    cbthick - blade wing thickness                         (input)
 - -     trspan - blade tie rod span (cm)                      (input)
 - -    trthick - blade tie rod thickness (cm)                 (input)
 - -    wsthick - blade wing thickness (cm)                    (input)
 - -   cblength - active absorber length (cm)                  (input)
 - -        ncs - number of central stiffeners in control      (input)
 - -              blade wing
 - -      csoff - blade central stiffener offset (cm)          (input)
 - -    cswidth - blade central stiffener width (cm)           (input)
 - -     cbpmat - material indentifier for control blade poi-  (input)
 - -              sen
 - -      atmat - material identifier for absorber tube mater- (input)
 - -              ial
 - -     cbsmat - material identifier for blade sheath mater-  (input)
 - -              ial
 - -    cbtrmat - material identifier for blade tie rod mat-   (input)
 - -              erial
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - -
 - - Character Variable(s)
 - -  buffer - string variable used to load entire lines for print
 - -   sbuff - short string variable used to load a segment of a line
*/
  char sbuff[5];
  char buffer[133];
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Summarize NAMELIST Input Directives - - - - - - - - - - - - - - - - */
  header();
  lines(5);
  fprintf(nout
```

```
 ,"0Dataset Edit -- Values from Blade Geometry Dataset\n");
fprintf(nout,"0 Variable     Value    Definition\n");
fprintf(nout,"\n");
lines(1);
sprintf(buffer,"    NTUBE     %6i",ntube);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Number of Absorber Tubes in Blade\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"   CBSPAN     %6.2f",cbspan);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Control Blade Span (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"     ATID     %6.2f",atid);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Absorber Tube Inner Diameter (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"     ATOD     %6.2f",atod);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Absorber Tube Outer Diameter (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"  CBTHICK     %6.2f",cbthick);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Wing Thickness (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"   TRSPAN     %6.2f",trspan);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Tie Rod Span\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"  TRTHICK     %6.2f",trthick);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Tie Rod Thickness (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"  WSTHICK     %6.2f",wsthick);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Wing Thickness (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer," CBLENGTH     %6.2f",cblength);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Active Absorber Length (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"   CBPMAT      %s",cbpmat);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Blade Poison Material\n");
fprintf(nout,buffer);
lines(1);
```

```
  sprintf(buffer,"    ATMAT      %s",atmat);
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer,"Absorber Tube Material\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"   CBSMAT      %s",cbsmat);
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer,"Blade Sheath Material\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer," CBTRMAT      %s",cbtrmat);
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer,"Blade Tie Rod Material\n");
  fprintf(nout,buffer);
  switch (ncs){
  case 0:
    lines(1);
    sprintf(buffer,"    CSOFF     %6.2f",csoff);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Central Stiffener Offset (cm)\n");
    fprintf(nout,buffer);
    lines(1);
    sprintf(buffer," CSWIDTH     %6.2f",cswidth);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Central Stiffener Width (cm)\n");
    fprintf(nout,buffer);
    break;
  case 1:
    break;
  default:
    lines(3);
    fprintf(nout
     ,"0***  F a t a l  E r r o r  ***  -- Function");
    fprintf(nout," bladedb_edt:\n");
    fprintf(nout
     ," NCS must be either Zero or Unity,");
    fprintf(nout," Value Encountered = %i\n",ncs);
    abort();}
 lines(1);
 fprintf(nout,"\n");
 return;
}
```

Function coredb_edt

```
#include <stdio.h>
#include <string.h>

void coredb_edt(int nrow,int ncol,int nrowb,int ncolb,float apitch
 ,float vod,float vthick,float sod,float sthick,float tutpr
 ,float tcgr,float bltpr,float bcpr,int *incore_loc,char mvessel[]
 ,char mshroud[],char mtg[],char mcp[],char migt[],float dtod
 ,float dtid){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * c o r e d b _ e d t  *  Edits Contents of Core Geometry Data
 - -                              set
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -       nrow - number of rows in core map for the whole core (input)
 - -       ncol - number of columns in core map for the whole   (input)
 - -              core
 - -      nrowb - number of rows of control cell locations in   (input)
 - -              the core
 - -      ncolb - number of columnes of control cell locations  (input)
 - -              in the core
 - -     apitch - fuel assembly pitch                           (input)
 - -        sod - shroud outer radius                           (input)
 - -     sthick - shroud thickness                              (input)
 - -        vod - pressure vessel outer radius                  (input)
 - -     vthick - pressure vessle thickness                     (input)
 - -      tutpr - Top of Upper Tie Plate Region                 (input)
 - -       tcgr - Top of Core Grid Region                       (input)
 - -      bltpr - Bottom of Lower Tie Plate Region              (input)
 - -       bcpr - Bottom of Core Plate Region                   (input)
 - - incore_loc - positions for incore instrumentation          (input)
 - -    mvessel - vessel material identifier                    (input)
 - -    mshroud - core shroud material identifier               (input)
 - -        mtg - core top guide region material identifier     (input)
 - -        mcp - core plate region material identifier         (input)
 - -       migt - incore guide tube material identifier         (input)
 - -       dtod - outer diameter of incore guide tube           (input)
 - -       dtid - inner diameter of incore guide tube           (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Character Variable(s)
 - -  buffer - string variable used to load entire lines for print
 - -   sbuff - short string variable used to load a segment of a line
*/
  char sbuff[5];
  char buffer[133];
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Summarize NAMELIST Input Directives - - - - - - - - - - - - - - - - - */
  header();
```

```
lines(5);
fprintf(nout,"0Dataset Edit -- Values from Core Geometry Dataset\n");
fprintf(nout,"0Variable     Value    Definition\n");
fprintf(nout,"\n");
lines(2);
sprintf(buffer,"    NCOL     %6i",ncol);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Number of Columns of Fuel Assemblies in the Core\n");
fprintf(nout,buffer);
lines(2);
sprintf(buffer,"    NROW     %6i",nrow);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Number of Rows of Fuel Assemblies in the Core\n");
fprintf(nout,buffer);
lines(3);
sprintf(buffer,"   NCOLB     %6i",ncolb);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Number of Columns of Control Blades in Core\n");
fprintf(nout,buffer);
sprintf(buffer,"   NROWB     %6i",nrowb);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Number of Rows of Control Blades in Core\n");
fprintf(nout,buffer);
sprintf(buffer,"  APITCH     %6.2f",apitch);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Fuel Assembly Pitch\n");
fprintf(nout,buffer);
sprintf(buffer,"     VOD     %6.2f",vod);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Vessel Outer Diameter (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"  VTHICK     %6.2f",vthick);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Vessel Thickness (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"     SOD     %6.2f",sod);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Core Shroud Outer Diameter (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"  STHICK     %6.2f",sthick);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Core Shroud Thickness (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"   TUTPR     %6.2f",tutpr);
bufferpad(buffer,strlen(buffer),22);
strcat(buffer,"Top of the Upper Tie Plate Region (cm)\n");
fprintf(nout,buffer);
lines(1);
sprintf(buffer,"    TCGR     %6.2f",tcgr);
bufferpad(buffer,strlen(buffer),22);
```

```
  strcat(buffer,"Top of the Core Grid Region (cm)\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"   BLTPR     %6.2f",bltpr);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer,"Bottom of the Lower Tie Plate Region (cm)\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"    BCPR     %6.2f",bcpr);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer
   ,"Bottom of the Fuel Support/Core Plate Region (cm)\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"    DTOD     %6.2f",dtod);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer
   ,"Outer Diameter of Incore Guide Tube (cm)\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"    DTID     %6.2f",dtid);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer
   ,"Inner Diameter of Incore Guide Tube (cm)\n");
  fprintf(nout,buffer);
/* - Material Identifiers */
  lines(1);
  sprintf(buffer," MVESSEL      %s",mvessel);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer
   ,"Material Identifier for Vessel\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer," MSHROUD      %s",mshroud);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer
   ,"Material Identifier for Core Shroud\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"     MTG      %s",mtg);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer
   ,"Material Identifier for Core Top Guide Region\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"     MCP      %s",mcp);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer
   ,"Material Identifier for Core Plate Region\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"    MIGT      %s",migt);
  bufferpad(buffer,strlen(buffer),22);
  strcat(buffer
```

```
   ,"Material Identifier for Incore Guide Tube\n");
  fprintf(nout,buffer);
/* - Edit Incore Instrumentation Location Map */
  lines(5+nrowb);
  strcpy(buffer,"");
  bufferpad(buffer,0,132);
  fprintf(nout,"0Incore Instrumentation Location Map\n");
  fprintf(nout," (1 - indicates instrumentation dry tube present)\n");
  fprintf(nout,"\n");
  { short int i,j;
    int *p = incore_loc;
    fprintf(nout,"   ");
    for(i = 1; i <= ncolb; i++){
       short int ib;
       ib = (2*i)-1;
       fprintf(nout,"  %2i",ib);
    }
    fprintf(nout,"\n");
    fprintf(nout,"\n");
    for(j = 0; j < nrowb; j++){
      short int jb;
      jb = (2*(j+1))-1;
      sprintf(buffer," %2i",jb);
      for(i = 0; i < ncolb; i++){
        if(*p == 0)
          strncat(buffer,"    ",4);
        else{
          sprintf(sbuff," %3i",*p);
          strncat(buffer,sbuff,4);
        }
        p++;
      }
      fprintf(nout,buffer);
      fprintf(nout,"\n");
    }
  }
 return;
}
```

Function edit_ct

```
#include <stdio.h>
#include <string.h>

void edit_ct(int nlatticm,int *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  e d i t _ c t  *  Edits Contents of Correspondence Table, which
 - -                      Relates Lattice Geometry Indices to Lattice
 - -                      Material Indices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -   nlatticm - Number of Unique Lattice Material Indices      (input)
 - -          p - Pointer to Correspondence Table               (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
```

```
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  n - index for loop
 - - p2 - pointer to second member of correspondence table
*/
  int n, *p2;
/* - File Variable(s)
 - -  nout - output stream
*/
  extern FILE *nout;
/* - Edit Correspondence Table - - - - - - - - - - - - - - - - - - - - - - - */
  lines(7);
  fprintf(nout,"0Correspondence Table for Lattice Material and Geometry");
  fprintf(nout," Indices\n");
  fprintf(nout,"0Index Material Geometry\n\n");
  p2 = p+1;
  for( n = 1; n <= nlatticm; n++){
    if(n > 7) lines(1);
    fprintf(nout," %3i   %6i   %6i\n",n,*p,*p2);
    p++; p++;
    p2 = p+1;}
}
```

Function edit_spacer

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

void edit_spacer(int nbundlg,int *ptr_n_spacer
 ,float *ptr_spacer_location,ascii_string *ptr_spacer_material){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  e d i t _ s p a c e r  *  Edits Input Variables Defining
 - -                               Fuel Assembly Spacers
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -    nbundlg - number of fuel assembly geometrical types    (input)
 - - ptr_n_spacer
 - -            - number of spacers in each fuel assembly type (input)
 - - ptr_spacer_location
 - -            - pointer to array containing location of spa- (input)
 - -              cers for each fuel assembly type
 - - ptr_spacer_material
 - -            - pointer to array containing spacer material  (input)
 - -              mnemonic for each fue assembly type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -      i - utility index variable
 - -      j - utility index variable
 - -  ptr_n - pointer to integer vector
*/
  short int i,j;
```

```
  int *ptr_n = ptr_n_spacer;
/* - Float Variable(s)
 - - ptr_f - pointer to floating point array
*/
  float *ptr_f = ptr_spacer_location;
/* - Character Variable(s)
 - - ptr_c - pointer to ascii_string vector
*/
  ascii_string *ptr_c = ptr_spacer_material;
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Summarize NAMELIST Input Directives - - - - - - - - - - - - - - - - */
  header();
  lines(5+nbundlg);
  fprintf(nout
   ,"0Fuel Assembly Spacer Information\n");
/* - Number of Spacers for Each Unique Fuel Assembly Geometry */
  fprintf(nout,"0Index  Number of Spacers  Material Mnemonic\n\n");
  for(i = 1;i <= nbundlg;i++){
    fprintf(nout," %3i          %2i                  %s\n"
     ,i,*ptr_n,*ptr_c);
    ptr_n++;
    ptr_c++;}
/* - Spacer Coordinates for Each Type */
  lines(3+nbundlg);
  fprintf(nout,"0Index Axial Distance from BAF (cm)\n\n");
  ptr_n = ptr_n_spacer;
  for(i = 1;i <= nbundlg;i++){
    fprintf(nout," %3i",i);
    for(j = 1;j <= *ptr_n;j++){
      fprintf(nout," %7.3f",*ptr_f);
      *ptr_f++;
    }
    fprintf(nout,"\n");
    ptr_n++;
  }
}
```

Function editin

```
#include<stdio.h>
#include<string.h>
typedef char ascii_string[133];

void editin(char core_db[],char lprefix[],char fprefix[],int naxial
 ,int naxpl,float afl,int nrowp,int ncolp,int nrowbp,int ncolbp
 ,int nbundlg,int nbundlm,int *gmap,int *mmap,int *lgvect,int *lmvect
 ,int *bladep,int nlatticg,int nlatticm,ascii_string *lgdsnam
 ,ascii_string *lmdsnam,char core_mtls[],int core_f,char blade_db[]
 ,float rho,float rhobyp,float tempk,char mutp[],char mltp[]){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * e d i t i n  *  Edits Input Directives
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
 - - Argument(s):
 - -    core_db - file name for core geometry dataset          (input)
 - -    lprefix - file specification prefix for lattice geo-   (input)
 - -              metry datasets
 - -    fprefix - file specification prefix for fuel material  (input)
 - -              intermediate datasets
 - -     naxial - number of axial nodes in problem             (input)
 - -      naxpl - number of axial nodes in core + 1            (input)
 - -        afl - active fuel length                           (input)
 - -      nrowp - number of rows of fuel in the problem        (input)
 - -      ncolp - number of columns of fuel in the problem     (input)
 - -     ncolbp - number of control blade rows in the problem  (input)
 - -    nbundlg - number of distinct fuel assembly geometries  (input)
 - -              in the core
 - -    nbundlm - number of distinct fuel assembly material    (input)
 - -              types in the core
 - -       gmap - map for fuel assembly geometical indices     (input)
 - -       mmap - map for fuel assembly material indices       (input)
 - -     lgvect - vectors mapping fuel lattice indices to fuel (input)
 - -              assembly indices for geometries
 - -     lmvect - vectors mapping fuel lattice material indices (input)
 - -              to fuel assembly material indices
 - -     bladep - map of control blade positions               (input)
 - -   nlatticg - number of unique lattice geometries          (input)
 - -   nlatticm - number of unique lattice materials            (input)
 - -   lgsdsnam - pointer to character strings holding names of (input)
 - -              lattices geometry datasets
 - -   lmsdsnam - pointer to character strings holding names of (input)
 - -              fuel material intermediate datasets
 - -  core_mtls - file description for core materials dataset  (input)
 - -     core_f - fraction of core modeled                     (input)
 - -              (1 - full, 2 - half, 4 - quarter)
 - -   blade_db - control blade dataset for problem            (input)
 - -        rho - input density of inchannel moderator (g/cc) (input)
 - -     rhobyp - input density of bypass moderator (g/cc)     (input)
 - -      tempk - problem temperature (Kelvin)                 (input)
 - -       mutp - material identifier for Upper Tie Plate      (input)
 - -              Region
 - -       mltp - material identifier for Lower Tie Plate      (input)
 - -              Region
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Character Variable(s)
 - -  buffer - string variable used to load entire lines for print
 - -   sbuff - short string variable used to load a segment of a line
*/
  char sbuff[5];
  char buffer[133];
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Summarize NAMELIST Input Directives - - - - - - - - - - - - - - - - */
```

```
header();
lines(5);
fprintf(nout,"0Input Edit -- Values from Input Deck Card Images\n");
fprintf(nout,"0 Variable    Value    Definition\n");
fprintf(nout,"\n");
lines(2);
strcpy(buffer,"   CORE_DB");
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Location of Core Geometry Database\n");
fprintf(nout,buffer);
fprintf(nout,"                %s\n",core_db);
lines(2);
strcpy(buffer," CORE_MTLS");
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Location of Core Materials Database\n");
fprintf(nout,buffer);
fprintf(nout,"                %s\n",core_mtls);
lines(2);
strcpy(buffer,"   LPREFIX");
bufferpad(buffer,strlen(buffer),23);
strcat(buffer
 ,"File Specification for Lattice Geometry Datasets\n");
fprintf(nout,buffer);
fprintf(nout,"                %s\n",lprefix);
lines(2);
strcpy(buffer,"   FPREFIX");
bufferpad(buffer,strlen(buffer),23);
strcat(buffer
 ,"File Specification for Lattice Material Datasets\n");
fprintf(nout,buffer);
strcpy(buffer,"  BLADE_DB");
bufferpad(buffer,strlen(buffer),23);
strcat(buffer
 ,"File Specification for Blade Geometry Datasets\n");
fprintf(nout,buffer);
fprintf(nout,"                %s\n",fprefix);
lines(4);
sprintf(buffer,"     NROWP    %6i",nrowp);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Number of Rows in Problem Core Map\n");
fprintf(nout,buffer);
sprintf(buffer,"     NCOLP    %6i",ncolp);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Number of Columns in Problem Core Map\n");
fprintf(nout,buffer);
sprintf(buffer,"    NAXIAL    %6i",naxial);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Number of Axial Nodes\n");
fprintf(nout,buffer);
sprintf(buffer,"      AFL    %7.2f",afl);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Active Core Height (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"   CORE_F     %6i",core_f);
```

```
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer,"Fraction of Core Modeled:");
  switch (core_f){
    case 1:
      strcat(buffer," full-core problem\n"); break;
    case 2:
      strcat(buffer," half-core problem\n"); break;
    case 4:
      strcat(buffer," quarter-core problem\n"); break;
    default:
      lines(3);
      fprintf(nout,"0***  F a t a l  E r r o r -- Function");
      fprintf(nout," editin:  Illegal Input for core_f\n");
      fprintf(nout," Value Encountered = %i\n");
      abort();
  }
  fprintf(nout,buffer);
  lines(2);
  sprintf(buffer,"      RHO    %7.5f",rho);
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer,"Inchannel Density (g/cc)\n");
  fprintf(nout,buffer);
  sprintf(buffer,"   RHOBYP    %7.5f",rhobyp);
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer,"Bypass Density (g/cc)\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"    TEMPK    %7.2f",tempk);
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer,"Problem Temperature (Kelvin)\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"     MUTP     %s",mutp);
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer
   ,"Material Identifier for Upper Tie Plate Region\n");
  fprintf(nout,buffer);
  lines(1);
  sprintf(buffer,"     MLTP     %s",mltp);
  bufferpad(buffer,strlen(buffer),23);
  strcat(buffer
   ,"Material Identifier for Lower Tie Plate Region\n");
  fprintf(nout,buffer);
/* - Edit Fuel Assembly Geometry Assignment Map */
  lines(4+nrowp);
  fprintf(nout,"0Fuel Assembly Geometry Index Map\n");
  fprintf(nout,"\n");
  { short int i,j;
    int *p = gmap;
    fprintf(nout,"   ");
    for(i = 1; i <= ncolp; i++)
     fprintf(nout,"  %2i",i);
    fprintf(nout,"\n");
    fprintf(nout,"\n");
```

```
    for(j = 0; j < nrowp; j++){
      sprintf(buffer," %2i",(j+1));
      for(i = 0; i < ncolp; i++){
        if(*p == 0)
          strncat(buffer,"    ",4);
        else{
          sprintf(sbuff," %3i",*p);
          strncat(buffer,sbuff,4);
        }
        p++;
      }
      fprintf(nout,buffer);
      fprintf(nout,"\n");
    }
  }
/* - Edit Lattice Geometry Assignment Vectors */
  lines(7);
  fprintf(nout,"0Lattice Geometry Loading Vectors\n");
  fprintf(nout,"0Fuel Assembly Index/Lattice Index\n");
  {  short int i,j;
     int *p = lgvect;
     fprintf(nout," bundle");
     for(i = 1; i <= naxial; i++)
       fprintf(nout,"%5i",i);
     fprintf(nout,"\n");
     fprintf(nout,"\n");
     for(j = 0; j < nbundlg; j++){
       fprintf(nout," %5i ",*p);
       p++;
       for(i = 1; i < naxpl; i++){
         fprintf(nout,"%5i",*p);
         p++;
       }
       if(i > 1) lines(1);
       fprintf(nout,"\n");
     }
  }
/* - Edit Lattice Geometry Dataset File Specifications */
  lines(5);
  strcpy(buffer,"");
  bufferpad(buffer,0,132);
  fprintf(nout,"0Lattice Geometry Datasets\n");
  fprintf(nout,"\n");
  {  short int i;
     ascii_string *p = lgdsnam;
     for(i = 0; i < nlatticg; i++, p++){
       sprintf(buffer," %2i ",i+1);
       strncat(buffer,*p,132);
       if(i > 4) lines(1);
       fprintf(nout,buffer);
       fprintf(nout,"\n");}
  }
/* - Edit Fuel Assembly Material Assignment Map */
  lines(4+nrowp);
```

```
  strcpy(buffer,"");
  bufferpad(buffer,0,132);
  fprintf(nout,"0Fuel Assembly Material Index Map\n");
  fprintf(nout,"\n");
  { short int i,j;
    int *p = mmap;
    fprintf(nout,"   ");
    for(i = 1; i <= ncolp; i++)
     fprintf(nout,"  %2i",i);
    fprintf(nout,"\n");
    fprintf(nout,"\n");
    for(j = 0; j < nrowp; j++){
      sprintf(buffer," %2i",(j+1));
      for(i = 0; i < ncolp; i++){
        if(*p == 0)
          strncat(buffer,"    ",4);
        else{
          sprintf(sbuff," %3i",*p);
          strncat(buffer,sbuff,4);
        }
        p++;
      }
      fprintf(nout,buffer);
      fprintf(nout,"\n");
    }
  }
/* - Edit Lattice Material Assignment Vectors */
  lines(7);
  fprintf(nout,"0Lattice Material Loading Vectors\n");
  fprintf(nout,"0Fuel Assembly Index/Lattice Index\n");
  {  short int i,j;
     int *p = lmvect;
     fprintf(nout," bundle");
     for(i = 1; i <= naxial; i++)
       fprintf(nout,"%5i",i);
     fprintf(nout,"\n");
     fprintf(nout,"\n");
     for(j = 0; j < nbundlm; j++){
       fprintf(nout," %5i ",*p);
       p++;
       for(i = 1; i < naxpl; i++){
         fprintf(nout,"%5i",*p);
         p++;
       }
       if(i > 1) lines(1);
       fprintf(nout,"\n");
     }
  }
/* - Edit Lattice Material Intermediate Dataset File Specifications */
  lines(5);
  strcpy(buffer,"");
  bufferpad(buffer,0,132);
  fprintf(nout,"0Lattice Material Intermediate Datasets\n");
  fprintf(nout,"\n");
```

```
  {  short int i;
     ascii_string *p = lmdsnam;
     for(i = 0; i < nlatticm; i++, p++){
       sprintf(buffer," %2i ",i+1);
       strncat(buffer,*p,132);
       if(i > 4) lines(1);
       fprintf(nout,buffer);
       fprintf(nout,"\n");}
  }
/* - Edit Blade Positioning Map */
  lines(4+nrowbp);
  strcpy(buffer,"");
  bufferpad(buffer,0,132);
  fprintf(nout,"0Control Blade Positioning Map\n");
  fprintf(nout,"\n");
  { short int i,j;
    int *p = bladep;
    fprintf(nout,"   ");
    for(i = 1; i <= ncolbp; i++){
       short int ib;
       ib = (2*i)-1;
       fprintf(nout,"  %2i",ib);
    }
    fprintf(nout,"\n");
    fprintf(nout,"\n");
    for(j = 0; j < nrowbp; j++){
      short int jb;
      jb = (2*(j+1))-1;
      sprintf(buffer," %2i",jb);
      for(i = 0; i < ncolbp; i++){
        if((*p == 48) | (*p == -1))
          strncat(buffer,"    ",4);
        else{
          sprintf(sbuff," %3i",*p);
          strncat(buffer,sbuff,4);
        }
        p++;
      }
      fprintf(nout,buffer);
      fprintf(nout,"\n");
    }
  }
 return;
}
```

Function fgds_edt

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct fuel_gometry{
                              ascii_string gds_name;
                              int latdim;
                              int nwr;
```

```
                                float cthick;
                                float asin;
                                float wgap;
                                float ngap;
                                float cradius;
                                float fsrd;
                                float cfsrd;
                                float rpitch;
                                float cod;
                                float cld;
                                float pod;
                                char frcmat[6];
                                char fcmat[6];
                              } fg_list;

  void fgds_edt(int nlatticg,char lprefix[],ascii_string *lgdsnam
   ,fg_list *ptr_lg_ds){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - fgds_edt - edits contents of fuel geometry datasets
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -   lprefix - prefix for lattice geometry datasets            (input)
 - -   lgdsnam - file descriptions for lattice geometry datasets (input)
 - - ptr_lg_ds - pointer to first element in list of fuel geo-   (input)
 - -             datasets
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   i - loop control variable
*/
  short int i;
/* - Character Variable(s)
 - -     fn - filename for lattice geometry dataset
 - - buffer - string used to manage output
*/
  ascii_string fn, buffer;
/* - FILE Variable(s)
 - -  nout - stream for output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  p - pointer to current element in list of fuel geometry datasets
 - -  l - pointer to current element in list of lattice gometry dataset
 - -      names
*/
  fg_list *p = ptr_lg_ds;
  ascii_string *l = lgdsnam;
/* - Loop Over all the Lattice Geometry Types - - - - - - - - - - - - - - - */
  for( i = 1; i <= nlatticg; i++,p++,l++){
    lines(20);
    strcpy(fn,lprefix);
    strcat(fn,*l);
```

```
fprintf(nout,"0Values from Lattice Geometry Dataset:\n  %s\n",fn);
fprintf(nout,"0 Variable     Value    Definition\n\n");
sprintf(buffer,"    LATDIM    %6i",p->latdim);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Lattice Dimensionality\n");
fprintf(nout,buffer);
sprintf(buffer,"       NWR    %6i",p->nwr);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Number of Water Rods\n");
fprintf(nout,buffer);
sprintf(buffer,"    CTHICK     %5.2f",p->cthick);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Channel Thickness (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"      ASIN     %5.2f",p->asin);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Channel Interior Span (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"      WGAP     %5.2f",p->wgap);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Wide Gap Half-thickness (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"      NGAP     %5.2f",p->ngap);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Narrow Gap Half-thickness (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"   CRADIUS     %5.2f",p->cradius);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Channel Corner Inner Radius (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"      FSRD     %5.2f",p->fsrd);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Clad Surface to Clad Surface Separation (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"     CFSRD     %5.2f",p->cfsrd);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer
 ,"Clad Surface to Inner Channel Surface Separation (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"    RPITCH     %5.2f",p->rpitch);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Fuel Rod Pitch (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"       COD     %5.2f",p->cod);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Fuel Rod Cladding Outer Diameter (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"       CLD     %5.2f",p->cld);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Fuel Rod Cladding Thickness (cm)\n");
fprintf(nout,buffer);
sprintf(buffer,"       POD     %5.2f",p->pod);
bufferpad(buffer,strlen(buffer),23);
strcat(buffer,"Fuel Pellet Outer Diameter (cm)\n");
```

```
    fprintf(nout,buffer);
    sprintf(buffer,"    FRCMAT     %s",p->frcmat);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Fuel Rod Cladding Material Identifier\n");
    fprintf(nout,buffer);
    sprintf(buffer,"     FCMAT     %s",p->fcmat);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Fuel Channel Material Identifier\n");
    fprintf(nout,buffer);
  }
}
```

2.5. Deck Generation Routines

Function add_cell

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

void bufferpad(char[],int,int);

void add_cell(char label[],char material[],FILE *lu8,int *ncell
 ,int index,float density,int *nuniverse){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  a d d _ c e l l  *  adds cell definition to MCNP model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      label - Label for Cell                                  (input)
 - -   material - Label for Cell Material                         (input)
 - -        lu8 - Pointer to Scratch File Stream for Cell Def-   (input)
 - -              initions
 - -      ncell - Cell Number                                     (input)
 - -      index - Index for Cell Material                         (input)
 - -    density - Density for Cell Material                       (input)
 - -  nuniverse - Universe for Cell                               (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Character Variable(s)
 - -   buffer - buffer for processing lines of output
 - -  ptr_buf - pointer to buffer
*/
  ascii_string buffer;
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Write Cell Definition to MCNP Input Deck - - - - - - - - - - - - -  */
  (*ncell)++;
  fprintf(lu8,"c     %s\n",label);
  if(index != 0)
    fprintf(lu8,"%5i %5i %10.4E",*ncell,index,density);
  else
    fprintf(lu8,"%5i %5i",*ncell,index);
/* - Write Cell Definition to Output File - - - - - - - - - - - - - - -  */
  sprintf(buffer," %5i",*ncell);
  bufferpad(buffer,strlen(buffer),(30-strlen(material)));
  ptr_buf = buffer+(30-strlen(material));
  sprintf(ptr_buf,"%s/%5i/%10.4E",material,index,density);
  if(*nuniverse != 0){
    ptr_buf = buffer+49*sizeof(char);
    bufferpad(buffer,strlen(buffer),49);
    sprintf(ptr_buf,"%4i",*nuniverse);}
```

```
  ptr_buf = buffer+59*sizeof(char);
  bufferpad(buffer,strlen(buffer),59);
  sprintf(ptr_buf,"%s\n",label);
  lines(1);
  fprintf(nout,buffer);
}
```

Function add_like_but

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

void bufferpad(char[],int,int);

void add_like_but(char label[],char material[],FILE *lu8,int *ncell
 ,char but_string[],int ncref,int *nuniverse){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  a d d _ c e l l  *  adds cell definition to MCNP model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      label - Label for Cell                                  (input)
 - -   material - Label for Cell Material                         (input)
 - -        lu8 - Pointer to Scratch File Stream for Cell Def-   (input)
 - -              initions
 - -      ncell - Cell Number                                     (input)
 - - but_string - String Variable Containing Exceptions to        (input)
 - -              Cloned Cell
 - -      ncref - Index of Cell to Clone                          (input)
 - -  nuniverse - Universe for Cell                               (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Character Variable(s)
 - -   buffer - buffer for processing lines of output
 - -  ptr_buf - pointer to buffer
*/
  ascii_string buffer;
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Write Cell Definition to MCNP Input Deck - - - - - - - - - - - - -  */
  (*ncell)++;
  fprintf(lu8,"c     %s\n",label);
  fprintf(lu8,"%5i like %i but \n",*ncell,ncref);
  if(strlen(but_string) != 0)
    fprintf(lu8,"      %s\n",but_string);
/* - Write Cell Definition to Output File - - - - - - - - - - - - - - -  */
  sprintf(buffer," %5i",*ncell);
  bufferpad(buffer,strlen(buffer),(30-strlen(material)));
  ptr_buf = buffer+(30-strlen(material));
  sprintf(ptr_buf,"%s",material,ncref);
```

```
  if(*nuniverse != 0){
    ptr_buf = buffer+49*sizeof(char);
    bufferpad(buffer,strlen(buffer),49);
    sprintf(ptr_buf,"%4i",*nuniverse);}
  ptr_buf = buffer+59*sizeof(char);
  bufferpad(buffer,strlen(buffer),59);
  sprintf(ptr_buf,"%s\n",label);
  lines(1);
  fprintf(nout,buffer);
  lines(1);
  fprintf(nout,"     %s\n",but_string);
}
```

Function add_material

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct s_material{
                          struct s_material *last;
                          int atomic_number;
                          int mass_number;
                          float weight_percentage;
                          char library_suffix[5];
                          struct s_material *next;
                         } ll_material;


void add_material(FILE *lu10,int *nmaterial,char label[],int n_entries
 ,ll_material *ptr_mtl) {
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  a d d _ m a t e r i a l  *  adds material definition
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -       lu10 - pointer to scratch stream for MCNP material    (input)
 - -              definitions
 - -  nmaterial - number of materials in MCNP problem              (i&o)
 - -      label - material identifier                             (input)
 - -  n_entries - number of entries in material definition        (input)
 - -              linked list
 - -    ptr_mtl - pointer to linked list for a specific material (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        i - loop index
 - -     nloc - location in string
 - - n_print - number of lines printed
 - -     nval - number of entries printed on a single line
*/
  int i, nloc, n_print = 1, nval = 0;
/* - Character Variable(s)
 - -    zaid - MCNP material identifier
 - -      cp - pointer to character string
```

```
 - - suffix - local variable for library suffix
*/
  char zaid[11];
  char *cp;
  char suffix[5];
/* - Structured Variable(s) */
  ll_material *pm;
/* - Copy Material Definition from Linked List - - - - - - - - - - - - - - */
  fprintf(lu10,"c     %s\n",label);
  pm = ptr_mtl;
  for(i = 1; i <= n_entries; i++){
    if(pm->weight_percentage != 0.0){
      cp = zaid;
      nloc = 3;
      if(pm->atomic_number < 100){
        sprintf(cp,"0");
        cp++;
        nloc--;}
      if(pm->atomic_number < 10){
        sprintf(cp,"0");
        cp++;
        nloc--;}
      sprintf(cp,"%i",pm->atomic_number);
      { short int ip;
        for(ip = nloc; ip > 0; ip--) cp++;
      }
      nloc = 3;
      if(pm->mass_number < 100){
        sprintf(cp,"0");
        cp++;
        nloc--;}
      if(pm->mass_number < 10){
        sprintf(cp,"0");
        cp++;
        nloc--;}
      sprintf(cp,"%i",pm->mass_number);
      { short int ip;
        for(ip = 1; ip <= nloc; ip++) cp++;
      }
      strncpy(suffix,pm->library_suffix,4);
      suffix[4] = '\0';
      sprintf(cp,"%s",suffix);
      if(n_print == 1){
        n_print++;
        nloc = 0;
        if(*nmaterial < 1000) nloc++;
        if(*nmaterial < 100) nloc++;
        if(*nmaterial < 10) nloc++;
        { short int ip;
          for(ip = 1; ip <= nloc; ip++) fprintf(lu10," ");
          fprintf(lu10,"m%i %s %10.4E",*nmaterial,zaid
           ,-(pm->weight_percentage));
          nval++;
        }
```

```
      }
      else{
        if(nval == 0) fprintf(lu10,"     ");
        fprintf(lu10," %s %10.4E",zaid
         ,-(pm->weight_percentage));
        n_print++;
        nval++;
        if(nval == 3){
          fprintf(lu10,"\n");
          nval = 0;
        }
      }
      pm = pm->next;
    }
  }
  if(nval != 0) fprintf(lu10,"\n");
}
```

Function add_surface

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;

surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);

void add_surface(int operation,ascii_string label,char mnemonic[4]
 ,FILE *lu8,FILE *lu9,int *index,ascii_string value
 ,surface_usage_list *ptr_surface_usage,int *nsurface){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  a d d _ s u r f a c e  *  adds surface to MCNP model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -  operation - operation to perform                          (input)
 - -              (0 - add new surface card,
 - -               1 - do not add new surface card)
 - -      label - Label Surface                                 (input)
 - -   mnemonic - MCNP mnemonic for surface (first edit only)   (input)
 - -        lu8 - Pointer to Scratch File Stream for Cell Def-  (input)
 - -              initions
 - -        lu9 - Pointer to Scratch File Stream for Surface    (input)
```

```
 - -              Definitions
 - -      index - Index for Surface                                (i|o)
 - -      value - String Containing Values for Defining Surface  (input)
 - - ptr_surface_usage
 - -            - Points to Surface Usage Linked List            (input)
 - -   nsurface - total number of surfaces                        (i|o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        multiplier - multiplier on index for cell card
 - -  equivalent_index - index for equivalent surface
*/
  int multiplier = 1.0, equivalent_index;
/* - Character Variable(s)
 - -           buffer - buffer for processing lines of output
 - -          ptr_buf - pointer to buffer
 - - equivalent_label - label for surface with identical definition
*/
  ascii_string buffer, equivalent_label = "";
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Structured Variable(s) - - - - - - - - - - - - - - - - - - - - - -
 - -  ptr_sl - pointer to surface usage list
*/
  surface_usage_list *ptr_sl;
/* - For New Surfaces,  Write to Surface Accumulation Scratch File */
  if(*index < 0) {
    multiplier = -1;
    *index = -(*index);}
  if(operation == 0){
/* - Search for Equivalent Surface that has already been Defined */
    search_surface_usage_list(2,"",&equivalent_index,value,mnemonic
     ,equivalent_label,ptr_surface_usage);
    ptr_sl = ptr_surface_usage;
    while(ptr_sl->next != NULL) ptr_sl = ptr_sl->next;
    if(equivalent_index == 0){
      (*nsurface)++;
      *index = *nsurface;}
    else
      *index = equivalent_index;
    ptr_sl = load_surface_usage_list(label,*index,value,mnemonic
     ,equivalent_label,ptr_sl);
    if(equivalent_index == 0){
      fprintf(lu9,"c     %s\n",label);
      fprintf(lu9,"%5i %s %s\n",*index,mnemonic,value);}
  }
/* - Write Surface Definition to Cell Card in MCNP Input Deck - - - -  */
  fprintf(lu8," %i",(multiplier*(*index)));
/* - Write to Edit */
  sprintf(buffer,"       %6i %s",(multiplier*(*index)),mnemonic);
```

```
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"%s\n",label);
  lines(1);
  fprintf(nout,buffer);
}
```

Function add_symmetry_surfaces

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;


void  search_surface_usage_list(int,char[],int *,char[],char[],char[]
 ,surface_usage_list *);
surface_usage_list *load_surface_usage_list(char[],int,char[],char[]
 ,char[],surface_usage_list *);

void add_symmetry_surfaces(int core_f,FILE *lu8,FILE *lu9
 ,surface_usage_list *ptr_surface_usage,int colons){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  a d d _ s y m m e t r y _ s u r f a c e s  *  adds symmetry sur-
 - -                                                  faces to MCNP
 - -                                                  model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     core_f - Flag for Fraction of Core in Model            (input)
 - -              (0 - full, 2 - half, 4 - quarter)
 - -        lu8 - Pointer to Scratch File Stream for Cell Def-  (input)
 - -              initions
 - -        lu9 - Pointer to Scratch File Stream for Surface    (input)
 - -              Definitions
 - - ptr_surface_usage
 - -            - Pointer to Surface Usage Linked List          (input)
 - -     colons - Flag to Indicate that colons should be added  (input)
 - -              before Surface Indices in Cell Definition
 - -              (0 - no
 - -               1 - yes)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  index - index to surface
```

```
*/
  int index;
/* - Character Variable(s)
 - -   buffer - buffer for processing lines of output
 - -    label - label for surface
 - -  ptr_buf - pointer to buffer
 - - mnemonic - MCNP mnemonic for surface
 - -    value - surface definition
*/
  char mnemonic[4];
  ascii_string buffer, label, value;
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_sl - pointer to surface usage list
*/
  surface_usage_list *ptr_sl;
/* - Write Surface Definition to Cell Card in MCNP Input Deck - - - -  */
  strcpy(label,"X-Z Plane");
  search_surface_usage_list(1,label,&index,"","","",ptr_surface_usage);
  strcpy(mnemonic,"");
  if(colons == 1) fprintf(lu8,":");
/* - Add Surface for First Occurrence */
  if(index == 0){
    ptr_sl = ptr_surface_usage;
    while(ptr_sl->next != NULL) ptr_sl = ptr_sl->next;
    index = (ptr_sl->index)+1;
    strcpy(mnemonic,"py");
    strcpy(value,"0.0");
    ptr_sl = load_surface_usage_list(label,index,value,mnemonic
     ,"",ptr_sl);
    fprintf(lu9,"c     %s\n",label);
    fprintf(lu9," *%5i 0.0\n",index);}
  fprintf(lu8," %i",index);
  sprintf(buffer,"       %6i %s",index,mnemonic);
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"%s\n",label);
  lines(1);
  fprintf(nout,buffer);
  if(core_f == 4){
    if(colons == 1) fprintf(lu8,":");
    strcpy(label,"Y-Z Plane");
    search_surface_usage_list(1,label,&index,"","","""
     ,ptr_surface_usage);
    strcpy(mnemonic,"");
    if(index == 0){
      ptr_sl = ptr_surface_usage;
      while(ptr_sl->next != NULL) ptr_sl = ptr_sl->next;
      index = (ptr_sl->index)+1;
      strcpy(mnemonic,"px");
```

```
      strcpy(value,"0.0");
      ptr_sl = load_surface_usage_list(label,index,value,mnemonic
       ,"",ptr_sl);
      fprintf(lu9,"c     %s\n",label);
      fprintf(lu9," *%5i 0.0\n",index);}
    fprintf(lu8," %i",(-index));
    sprintf(buffer,"       %6i %s",(-index),mnemonic);
    bufferpad(buffer,strlen(buffer),59);
    ptr_buf = buffer+59*sizeof(char);
    sprintf(ptr_buf,"%s\n",label);
    lines(1);
    fprintf(nout,buffer);
  }
}
```

Function augment_lattice_list

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct all{
                    struct all *last;
                    int basis_lattice_material_index;
                    int lattice_material_index;
                    struct all *next;
                   } augmented_lattice_list;


augmented_lattice_list *memory_lattice_list(int
 ,augmented_lattice_list *);

void augment_lattice_list(int *nlatticm,int *lmvect,int *lgvect
 ,int nlatticm_ref,int nlatticg,int nbundlm,int nbundlg,int naxial
 ,int *ptr_n_spacer,int *ptr_spacer_node
 ,augmented_lattice_list *additional_lattices
 ,int *ptr_correspondence_table){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * a u g m e n t _ l a t t i c e _ l i s t  *  adds lattices incor-
 - -                                                porating spacer grids
 - -                                                into material lattice
 - -                                                loading vectors
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -   nlatticm - total number of unique material lattice indices  (i&o)
 - -     lmvect - material lattice loading vectors                 (i&o)
 - -     lgvect - geometrical lattice loading vectors            (input)
 - -  nlatticm_ref
 - -            - total number of unique material lattice indi-  (input)
 - -              ces without augmentation for lattices incor-
 - -              porating spacers
 - -   nlatticg - total number of unique geometrical lattice in- (input)
 - -              dices
 - -    nbundlm - number of unique fuel assembly materials       (input)
```

```
 - -    nbundlg - number of unique fuel assembly geometries      (input)
 - -     naxial - number of axial nodes in problem               (input)
 - - ptr_n_spacer
 - -            - number of spacer grids associated with each    (input)
 - -              geometrically unique fuel assembly
 - - ptr_spacer_node
 - -            - nodal location of spacer grids for each geo-   (input)
 - -              metric fuel assembly type
 - - additional_lattices
 - -            - linked list of additional lattices generated  (output)
 - -              to accomodate spacer grids
 - - ptr_correspondence_table
 - -            - table providing relationship between lattice   (input)
 - -              material indices and lattice geometrical in-
 - -              dices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -              k - axial index for looping
 - -              n - uility loop variable
 - -          nglat - lattice geometry index
 - -          ngasm - fuel assembly geometry index
 - -            lmv - pointer to lattice material index list
 - -            lgv - pointer to lattice geometry index list
 - - spacer_present - flag to indicate whether a spacer grid is present
 - -                  at this location
*/
  short int k,n;
  int ngasm, spacer_present, nglat;
  int *lmv = lmvect, *lgv;
/* - Structured Variable(s)
 - -  ptr_al - pointer to additional lattice linked list
*/
  augmented_lattice_list *ptr_al = additional_lattices;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Sweep through Material Assembly Types - - - - - - - - - - - - - - - - */
  for(n = 1;n <= nbundlm;n++){
/* - Skip Fuel Assembly Index */
    lmv++;
/* - Sweep from Bottom of Assmbly to Top */
    for(k = 1;k <= naxial;k++,lmv++){
/* - Determine Corresponding Lattice Geometrical Index */
      { short int i;
        int *p = ptr_correspondence_table;
        while(*p != *lmv) {p++;p++;}
        nglat = *(p+1);
      }
/* - Find Fuel Assembly Geometrical Type of which the Lattice Geometry
 - - is a Member */
      lgv = lgvect;
```

```
      { short int i,bk,ns;
        int *ptr = ptr_n_spacer;
        int *ptr_n = ptr_spacer_node;
        ngasm = 0;
        for(i = 1;i <= nbundlg;i++)
          if(ngasm == 0){
/* - Skip Fuel Assembly Index */
            lgv++;
            for(bk = 1;bk <= naxial;bk++,lgv++)
              if(*lgv == nglat){
                ngasm = i;
                break;}
          }
          else break;
/* - Compare Nodes at which Spacer Grids are Located with Current Axial
 - - Node Index */
        spacer_present = 0;
        for(i = 1;i < ngasm;i++,ptr++)
          for(ns = 1;ns <= *ptr;ns++,ptr_n++);
        for(bk = 1;bk <= *ptr;bk++,ptr_n++)
          if(k == *ptr_n){
            spacer_present = 1;
            break;}
      }
/* - Determine whether this Lattice Incorporating a Spacer Grid has
 - - already been defined */
      if(spacer_present == 1){
        { short int found = 0, first = 0;
          augmented_lattice_list *ptr_next;
          ptr_al = additional_lattices;
          do{
              if(first != 0) ptr_al = ptr_al->next;
              first = 1;
              if(ptr_al->basis_lattice_material_index == *lmv){
                found = 1;
                *lmv = ptr_al->lattice_material_index;
                break;}
            }
          while(ptr_al->next != NULL);
/* - Spacer Grid not already Defined, Augment Lattice List */
          if(found == 0){
            ptr_al = additional_lattices;
/* - Check for First Entry in Linked List */
            if(ptr_al->lattice_material_index != NULL){
              while(ptr_al->next != NULL) ptr_al = ptr_al->next;
              ptr_next = memory_lattice_list(1,ptr_next);
              ptr_next->last = ptr_al;
              ptr_al->next = ptr_next;
              ptr_al = ptr_next;}
            (*nlatticm)++;
            ptr_al->basis_lattice_material_index = *lmv;
            ptr_al->lattice_material_index = *nlatticm;
            ptr_al->next = NULL;
            *lmv = *nlatticm;
```

```
            }
          }
        }
      }
    }
  }
```

Function build_assemblies

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;

void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void lines(int);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);

void build_assemblies(int *ncell,int *nuniverse,FILE *lu8,FILE *lu9
 ,surface_usage_list *ptr_surface_usage,float *ptr_nab,int **ptr_ufa
 ,int nbundlm,int *lmvect,int naxial,int *ptr_ufl,int *nsurface){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  b u i l d _ a s s e m b l i e s  *  builds fuel assemblies from
 - -                                        constituent lattices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  Argument(s):
 - -       ncell - number of cells in MCNP model                  (input)
 - -   nuniverse - number of "universes" in MCNP model            (input)
 - -         lu8 - stream pointer for scratch file for cell       (input)
 - -               definition accumulation
 - -         lu9 - stream pointer for scratch file for surface    (input)
 - -               definition accumulation
 - - ptr_surface_usage
 - -             - linked list for surfaces already defined in    (input)
 - -               the problem
 - -     ptr_nab - vector with axial corrdinates for the tops of (input)
 - -               nodes
 - -     ptr_ufa - vector with universe indices for unique fuel (output)
 - -               assembly types
 - -     nbundlm - number of unique fuel assembly types           (input)
 - -      lmvect - vector containing assignment of lattice mat-  (input)
 - -               erial indices to fuel assembly material indi-
 - -               ces
 - -      naxial - number of axial nodes                          (input)
 - -     ptr_ufl - vector with universe indices for unique lat-  (input)
 - -               tice types
 - -    nsurface - total number of surfaces                       (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
```

```
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -          n - loop variable
 - -          k - loop control for axial location
 - -      index - utility index variable
 - -        lmv - pointer to lmvect array
 - -          p - pointer to **ptr_ufa vector
*/
  short int n,k;
  int index, *lmv = lmvect;
  int *p = *ptr_ufa;
/* - Character Variable(s)
 - -    label - label for cell, surface or material
 - - mnemonic - mnemonic for surfaces
 - - material - name of material
 - -    value - string for defining surface definition
*/
  char mnemonic[4];
  ascii_string label, material, value;
/* - FILE Variable(s)
 - -  nout - pointer to code output file
*/
  extern FILE *nout;
/* - Loop Over Unqiue Fuel Assemblies in the Core - - - - - - - - - -  */
  for(n = 1;n <= nbundlm;n++){
    (*nuniverse)++;
    for(k = 1;k <= naxial;k++){
/* - Cells for Axial Nodes */
      sprintf(label,"Axial Node %i for Fuel Assembly %i",k,n);
      strcpy(material,"void");
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
/* - Surfaces for Node Boundaries */
      if(k != 1) {
        sprintf(label,"Top of Node %i",(k-1));
        search_surface_usage_list(1,label,&index,"","","""
         ,ptr_surface_usage);
        if(index == 0){
          lines(3);
          fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
          fprintf(nout," populate_control_cells --\n");
          fprintf(nout
           ," Surface not Found in Linked List, label = %s\n"
           ,label);}
        add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
         ,nsurface);}
      if(k != naxial){
        sprintf(label,"Top of Node %i",k);
        search_surface_usage_list(1,label,&index,"","","""
         ,ptr_surface_usage);
        if(index == 0){
          strcpy(mnemonic,"pz");
          { short int n;
            float *p = ptr_nab;
            for(n = 1;n < k;n++) p++;
```

```
            sprintf(value,"%10.4E",*p);
          }
          index = -1;
          add_surface(0,label,mnemonic,lu8,lu9,&index,value
           ,ptr_surface_usage,nsurface);}
        else{
          index = -index;
          add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
           ,nsurface);}
      }
      fprintf(lu8,"\n");
      { short int nn;
        int *l = lmv;
        int *p = ptr_ufl;
        l++;
        for(nn = 1;nn < k;nn++,l++) continue;
        for(nn = 1;nn < *l;nn++,p++) continue;
        fprintf(lu8,"      fill= %i\n",*p);
        fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
      }
    }
/*    **ptr_ufa = *nuniverse; */
    *p = *nuniverse;
    p++;
    for(k = 1;k <= (naxial+1);k++) lmv++;
  }
}
```

Function build_control_blade

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;
```

```
typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;


void add_material(FILE *,int *,char[],int,ll_material *);
void add_cell(char[],char[],FILE *,int *,int,float,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int,char[]
 ,char[],char[],surface_usage_list *);
ll_material *material_match(a_record *,char[],float *,int *);
int mchar(int *,char[]);
void lines(int);
void abort();
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void search_usage_list(int,char[],int *,usage_list *);
void rollup_llm(ll_material *p);

void build_control_blade(int ntube,float cbspan,float atid,float atod
 ,float cbthick,float trspan,float trthick,float wsthick,float cblength
 ,int ncs,float csoff,float cswidth,char cbpmat[],char atmat[]
 ,char cbsmat[],char cbtrmat[],usage_list *ptr_material_usage
 ,surface_usage_list *ptr_surface_usage,a_record *ptr_core_mtls
 ,FILE *lu8,FILE *lu9,FILE *lu10,int *nuniverse,int *nmaterial
 ,float bypass_density,int *ncell,int *ucb,int *nsurface
 ,int *ncell_tr){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  b u i l d _ c o n t r o l _ b l a d e  *  builds control blade
 - -                                              model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      ntube - number of absorber tubes in control blade      (input)
 - -     cbspan - control blade span                             (input)
 - -       atid - blade absorber tube inner diameter (cm)        (input)
 - -       atod - blade absorber tube outer diameter (cm)        (input)
 - -    cbthick - blade wing thickness                           (input)
 - -     trspan - blade tie rod span (cm)                        (input)
 - -    trthick - blade tie rod thickness (cm)                   (input)
 - -    wsthick - blade wing thickness (cm)                      (input)
 - -   cblength - active absorber length (cm)                    (input)
 - -        ncs - number of central stiffeners in control        (input)
 - -              blade wing
 - -      csoff - blade central stiffener offset (cm)            (input)
 - -    cswidth - blade central stiffener width (cm)             (input)
 - -     cbpmat - identifier for blade poison material           (input)
 - -      atmat - identifier for absorber tube material          (input)
 - -     cbsmat - identifier for blade sheath material           (input)
```

```
- -    cbtrmat - identifier for blade tie rod material          (input)
- - ptr_material_usage
- -            - pointer to linked list of material labels and  (input)
- -              indices
- - ptr_surface_usage
- -            - pointer to linked list of surface labels and   (input)
- -              indices
- - ptr_core_matls
- -            - pointer to linked list of core materials       (input)
- -        lu8 - pointer to stream containing scratch file for  (input)
- -              cell definitions
- -        lu9 - pointer to stream containing scratch file for  (input)
- -              surface definitions
- -       lu10 - pointer to stream containing scratch file for  (input)
- -              material definitions
- -  nuniverse - number of MCNP "universes"                      (i&o)
- -  nmaterial - number of MCNP materials                        (i&o)
- - bypass_density -
- -              density of water in bypass region              (input)
- -      ncell - number of MCNP cells (non-translated)           (i&o)
- -        ucb - universe number for control blade             (output)
- -   nsurface - total number of surfaces                        (i&o)
- -   ncell_tr - number of MCNP cell that are translated         (i&o)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- -
- - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
- - Integer Variable(s)
- -     sztmi - surface index for top of absorber column
- -     szbmi - surface index for bottom of absorber column
- -      spor - surface index for reference boron carbide outer radius
- -     sator - surface index for reference absorber tube outer radius
- -   sratmax - surface index for XAMX for reference absorber tube cell
- -   sratmin - surface index for XMIN for reference absorber tube cell
- -  sratymax - surface index for YMAX for reference absorber tube cell
- -  sratymin - surface index for YMIN for reference absorber tube cell
- -      trs2 - surface index for tie rod surface #2
- -      trs3 - surface index for tie rod surface #3
- -      trs4 - surface index for tie rod surface #4
- -      trs5 - surface index for tie rod surface #5
- -      trs6 - surface index for tie rod surface #6
- -      trs7 - surface index for tie rod surface #7
- -      trs8 - surface index for tie rod surface #8
- -    sambig - surface index for ambiguity surface for search corner
- -      scor - surface index for outer surface of sheath corner
- -      scir - surface index for inner surface of sheath corner
- -   sxminbw - surface index for XMIN for blade window
- -   sxmaxbw - surface index for XMAX for blade window
- -   syminbw - surface index for YMIN for blade window
- -   symaxbw - surface index for YMAX for blade window
- -      crat - cell index for reference absorber tube
- -       ctr - cell index for tie rod
- -        cs - cell index for control blade sheath
- -       cbw - cell index for wing window
- -      ccbn - cell index for wing window cloned to north location
```

```
 - -      ccbw - cell index for wing window cloned to west location
 - -      ccbs - cell index for wing window cloned to south location
 - -       cis - cell index for inside of sheath
 - -       uar - universe index for reference absorber tube
 - -        us - universe index for sheath and absorber tubes
 - -    ublade - universe index for entire blade
 - - n_entries - number of elements/isotopes of which a material is
 - -             composed
 - -     index - index of location in list
 - -      zero - the value zero
 - -     uwing - universe index for reference blade wing
*/
  int sztmi, szbmi, spor, sator, sratmax, sratmin, sratymax, sratymin
   , trs2, trs3, trs4, trs5, trs6, trs7, trs8, sambig, scor, scir
   , sxminbw, sxmaxbw, syminbw, symaxbw;
  int crat, ctr, cs, cbw, ccbn, ccbw, ccbs, cis;
  int uar, us, ublade, uwing;
  int n_entries = 0, index, zero = 0;
/* - Float Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -    ztmi - position of tip of blade absorber column when blade fully
 - -           inserted (143.75 inches for all BWR's)
 - -    zbmi - position of bottom of blade absorber column when blade
 - -           fully inserted
 - -     por - outer radius of boron carbide absorber mass
 - -    ator - outer radius of absorber tube
 - -   xcorn - Inner Surface of Sheath Corner
 - -     tfs - absorber tube field span
 - -   acmax - XMAX Surface for Reference Absorber Cell
 - -      x0 - X-coordinate for center of reference absorber tube
 - -  xambig - ambiguity surface for sheath corner
 - - density - density for material
 - -   rzero - the value zero
*/
  float ztmi = 365.125, zbmi, por, ator, xcorn, tfs, acmax, x0, xambig;
  float density, rzero = 0.0;
/* - Character Variable(s)
 - -   buffer - buffer for processing lines of output
 - -  ptr_buf - pointer to buffer
 - -    label - label for cell or surface
 - -    value - character representation of surface definition values
 - - mnemonic - identifier for MCNP surface
*/
  char mnemonic[4];
  ascii_string buffer, label, value;
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Structured Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  ptr_mtl - pointer to linked list for a specific material
 - -   ptr_ml - pointer to material usage list
*/
  ll_material *ptr_mtl;
```

```
  usage_list *ptr_ml;
/* - Compute Coordinates for Blade Generation - - - - - - - - - - - - - - */
  zbmi = ztmi-cblength;
  por = atid/2;
  ator = atod/2;
  xcorn = cbspan-wsthick;
  tfs = xcorn-trspan;
  acmax = (tfs/((float)(ntube/4)))+trspan;
  x0 = (acmax+trspan)/2;
  xambig = trspan+(((float)(ntube/4))-0.5)*(acmax-trspan);
/* - Create Blade Model in MCNP - - - - - - - - - - - - - - - - - - - - - - */
/* - Absorber - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells   */
  { int len = 6 , length;
    length = mchar(&len,cbpmat);
    cbpmat[length] = '\0';
  }
  search_usage_list(1,cbpmat,&index,ptr_material_usage);
  ptr_mtl = material_match(ptr_core_mtls,cbpmat,&density,&n_entries);
  if(index == 0){
    (*nmaterial)++;
    ptr_ml = ptr_material_usage;
    while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
    index = (ptr_ml->index)+1;
    ptr_ml = load_usage_list(cbpmat,index,ptr_ml);}
  else
    n_entries = 0;
  strcpy(label,"Boron Carbide in Reference Absorber Tube");
  (*nuniverse)++;
  uar = *nuniverse;
  add_cell(label,cbpmat,lu8,ncell,index,-density,nuniverse);
/* - Surfaces */
  strcpy(label,"Outer Radius of Absorber");
  strcpy(mnemonic,"c/z");
  sprintf(value,"%10.4E %10.4E %10.4E",x0,rzero,por);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0 u= %i\n",*nuniverse);
/* - Materials */
 if(n_entries != 0)
 { add_material(lu10,nmaterial,cbpmat,n_entries,ptr_mtl);
   rollup_llm(ptr_mtl);
   n_entries = 0;}
/* - Absorber Tube - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells   */
  { int len = 6 , length;
    length = mchar(&len,atmat);
    atmat[length] = '\0';
  }
  search_usage_list(1,atmat,&index,ptr_material_usage);
  ptr_mtl = material_match(ptr_core_mtls,atmat,&density,&n_entries);
  if(index == 0){
    (*nmaterial)++;
```

```
    ptr_ml = ptr_material_usage;
    while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
    index = (ptr_ml->index)+1;
    ptr_ml = load_usage_list(cbpmat,index,ptr_ml);}
  else
    n_entries = 0;
  strcpy(label,"Reference Absorber Tube");
  add_cell(label,cbpmat,lu8,ncell,index,-density,nuniverse);
/* - Surfaces */
  strcpy(label,"Outer Radius of Absorber Tube");
  strcpy(mnemonic,"c/z");
  sprintf(value,"%10.4E %10.4E %10.4E",x0,rzero,ator);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
  ,nsurface);
  strcpy(label,"Outer Radius of Absorber");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0 u= %i\n",*nuniverse);
/* - Materials */
 if(n_entries != 0)
 { add_material(lu10,nmaterial,atmat,n_entries,ptr_mtl);
   rollup_llm(ptr_mtl);
   n_entries = 0;}
/* - Water Outside of Reference Absorber Tube - - - - - - - - - - - - - - */
/* - Cells   */
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Material Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Water Outside Reference Absorber Tube");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,nuniverse);
/* - Surfaces */
  strcpy(label,"Outer Radius of Absorber Tube");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
```

```
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0 u= %i\n",*nuniverse);
/* - Materials */
/* - Bypass Water should already exist as a material */
/* - Window for Reference Absorber Tube - - - - - - - - - - - - - - - - - - - - */
/* - Cells   */
  (*nuniverse)++;
  us = *nuniverse;
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Material Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Window for Reference Absorber Tube Cell");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,nuniverse);
  crat = *ncell;
/* - Surfaces */
  strcpy(label,"XMAX for Reference Absorber Tube Cell");
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",acmax);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"XMIN for Reference Absorber Tube Cell");
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",trspan);
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"YMAX for Reference Absorber Tube Cell");
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4E",(trthick/2));
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"YMIN for Reference Absorber Tube Cell");
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4E",(-trthick/2));
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  fprintf(lu8
   ,"\n      imp:n=1.0 u= %i lat= 1 fill= 0:%i 0:0 0:0 %i %ir %i\n"
   ,*nuniverse,(ntube/4),uar,((ntube/4)-1),*nuniverse);
```

```
/* - Materials */
/* - Bypass Water should already exist as a material */
/* - Tie Rod - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells   */
  (*nuniverse)++;
  ublade = *nuniverse;
  { int len = 6 , length;
    length = mchar(&len,cbtrmat);
    cbtrmat[length] = '\0';
  }
  strcpy(label,cbtrmat);
  search_usage_list(1,label,&index,ptr_material_usage);
  ptr_mtl = material_match(ptr_core_mtls,cbtrmat,&density,&n_entries);
  if(index == 0){
    (*nmaterial)++;
    ptr_ml = ptr_material_usage;
    while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
    index = (ptr_ml->index)+1;
    ptr_ml = load_usage_list(cbpmat,index,ptr_ml);}
  else
    n_entries = 0;
  strcpy(label,"Tie Rod");
  add_cell(label,cbtrmat,lu8,ncell,index,-density
   ,nuniverse);
  ctr = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  strcpy(label,"Tie Rod Surface #6");
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4E",((trthick+wsthick)/2));
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"Tie Rod Surface #7");
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4E",(-(trthick+wsthick)/2));
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"XMIN for Reference Absorber Tube Cell");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  strcpy(label,"Tie Rod Surface #4");
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",(-trspan));
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"Top of Absorber Column");
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",ztmi);
```

```
index = -1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
strcpy(label,"Bottom of Absorber Column");
strcpy(mnemonic,"pz");
sprintf(value,"%10.4E",zbmi);
index = 1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
strcpy(label,"Tie Rod Surface #3");
strcpy(mnemonic,"px");
sprintf(value,"%10.4E",(-(trthick+wsthick)/2));
index = 1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
strcpy(label,"Tie Rod Surface #2");
strcpy(mnemonic,"px");
sprintf(value,"%10.4E",((trthick+wsthick)/2));
index = -1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
strcpy(label,"Tie Rod Surface #5");
strcpy(mnemonic,"py");
sprintf(value,"%10.4E",(-trspan));
index = 1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
strcpy(label,"Tie Rod Surface #8");
strcpy(mnemonic,"py");
sprintf(value,"%10.4E",trspan);
index = -1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
strcpy(label,"Top of Absorber Column");
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
strcpy(label,"Bottom of Absorber Column");
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
```

```
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")");
  fprintf(lu8,"\n      imp:n=1.0 u= %i\n",*nuniverse);
/* - Materials */
 if(n_entries != 0)
 { add_material(lu10,nmaterial,atmat,n_entries,ptr_mtl);
   rollup_llm(ptr_mtl);
   n_entries = 0;}
/* - Control Blade Sheath - - - - - - - - - - - - - - - - - - - - - - -   */
/* - Cells */
  (*nuniverse)++;
  uwing = *nuniverse;
  { int len = 6 , length;
    length = mchar(&len,cbsmat);
    cbsmat[length] = '\0';
  }
  strcpy(label,cbsmat);
  search_usage_list(1,label,&index,ptr_material_usage);
  ptr_mtl = material_match(ptr_core_mtls,cbsmat,&density,&n_entries);
  if(index == 0){
    (*nmaterial)++;
    ptr_ml = ptr_material_usage;
    while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
    index = (ptr_ml->index)+1;
    ptr_ml = load_usage_list(cbpmat,index,ptr_ml);}
  else
    n_entries = 0;
  strcpy(label,"Control Blade Sheath");
  add_cell(label,cbsmat,lu8,ncell,index,-density
   ,&us);
  cs = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  strcpy(label,"XMIN for Reference Absorber Tube Cell");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  strcpy(label,"Ambiguity Surface for Sheath Corner");
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",xambig);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
```

```
  ,nsurface);
 strcpy(label,"YMAX for Reference Absorber Tube Cell");
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
   fprintf(nout," build_control_blade --\n");
   fprintf(nout," Surface Not Found in Linked List label =");
   fprintf(nout," %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 strcpy(label,"Tie Rod Surface #6");
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
   fprintf(nout," build_control_blade --\n");
   fprintf(nout," Surface Not Found in Linked List label =");
   fprintf(nout," %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 strcpy(label,"Top of Absorber Column");
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
   fprintf(nout," build_control_blade --\n");
   fprintf(nout," Surface Not Found in Linked List label =");
   fprintf(nout," %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 strcpy(label,"Bottom of Absorber Column");
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
   fprintf(nout," build_control_blade --\n");
   fprintf(nout," Surface Not Found in Linked List label =");
   fprintf(nout," %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8,"):(");
 strcpy(label,"XMIN for Reference Absorber Tube Cell");
 search_surface_usage_list(1,label,&index,"","","""
```

```
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
strcpy(label,"Ambiguity Surface for Sheath Corner");
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
strcpy(label,"YMIN for Reference Absorber Tube Cell");
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
strcpy(label,"Tie Rod Surface #7");
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
strcpy(label,"Top of Absorber Column");
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
```

```
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  strcpy(label,"Bottom of Absorber Column");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n      (");
  strcpy(label,"Ambiguity Surface for Sheath Corner");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  strcpy(label,"Inner Surface for Sheath Corner");
  strcpy(mnemonic,"c/z");
  sprintf(value,"%10.4E %10.4E %10.4E",xambig,rzero,(trthick/2));
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"Outer Surface for Sheath Corner");
  strcpy(mnemonic,"c/z");
  sprintf(value,"%10.4E %10.4E %10.4E",xambig,rzero,((trthick+wsthick)/2));
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"Top of Absorber Column");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
```

```
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  strcpy(label,"Bottom of Absorber Column");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )");
  fprintf(lu8,"\n      imp:n=1.0 u= %i\n",uwing);
/* - Materials */
 if(n_entries != 0)
 { add_material(lu10,nmaterial,atmat,n_entries,ptr_mtl);
   rollup_llm(ptr_mtl);
   n_entries = 0;}
/* - Inside of Control Blade Sheath */
/* - Cell */
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Material Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Inside of Blade Wing");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&uwing);
  cis = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  strcpy(label,"XMIN for Sheath Interior");
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4f",(trspan+0.001));
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  strcpy(label,"YMIN for Sheath Interior");
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4f",(-(trthick/2)+0.001));
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  strcpy(label,"YMAX for Sheath Interior");
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4f",((trthick/2)-0.001));
```

```
index = -1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
strcpy(label,"Ambiguity Surface for Sheath Corner");
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
strcpy(label,"Top of Absorber Column");
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
strcpy(label,"Bottom of Absorber Column");
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8," ):( ");
strcpy(label,"Inner Surface for Sheath Corner");
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
  fprintf(nout," build_control_blade --\n");
  fprintf(nout," Surface Not Found in Linked List label =");
  fprintf(nout," %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
   ,nsurface);
  strcpy(label,"Top of Absorber Column");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  strcpy(label,"Bottom of Absorber Column");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Surface Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i fill= %i imp:n= 1.0\n",uwing,us);
/* - Region Outside of Blade Wing */
/* - Cell */
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Material Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Region Outside of Blade Wing");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&uwing);
/* - Surfaces */
  fprintf(lu8," #%i #%i",cs,cis);
  fprintf(lu8,"\n      imp:n=1.0 u= %i\n",uwing);
  sprintf(buffer,"           #%i #%i",cs,cis);
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"Definition formed by Complement Function\n");
  lines(1);
  fprintf(nout,buffer);
/* -  Window for Wing of Blade */
/* - Cell */
  strcpy(label,"Bypass Water");
```

```
  search_usage_list(1,label,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Material Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Window for Wing of Blade");
  add_cell(label,"Bypass Water",lu8,ncell_tr,index,-bypass_density
   ,nuniverse);
  cbw = *ncell_tr;
/* - Surfaces */
  strcpy(label,"XMIN for Blade Window");
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",(trspan+0.01));
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"XMAX for Blade Window");
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",(cbspan+0.01));
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"YMIN for Blade Window");
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4E",(-(trthick+wsthick+0.01)));
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"YMAX for Blade Window");
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4E",(trthick+wsthick+0.01));
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  fprintf(lu8," fill= %i",uwing);
  fprintf(lu8,"\n      imp:n=1.0 u= %i\n",ublade);
  lines(1);
  fprintf(nout,"     Filling Universe = %i\n",uwing);
/* - Materials */
/* - Bypass Water should already exist as a material */
/* - Wing Cloned to North Position - - - - - - - - - - - - - - - - - -  */
/* - Cell */
  (*ncell_tr)++;
  ccbn = *ncell_tr;
  strcpy(label,"Wing Cloned to North Position");
  fprintf(lu8,"c     %s\n",label);
  fprintf(lu8,"%5i",*ncell_tr);
  fprintf(lu8," like %i but",cbw);
  fprintf(lu8
   ," *trcl=( 0 0 0 90 0 90 180 90 90 90 90 0)\n");
  fprintf(lu8,"      u= %i\n",ublade);
```

```
  sprintf(buffer," %5i",*ncell_tr);
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"%s\n",label);
  lines(2);
  fprintf(nout,buffer);
  fprintf(nout,"      Cloning Directive:");
  fprintf(nout," like %i but",cbw);
  fprintf(nout
   ," *trcl=( 0 0 0 90 0 90 180 90 90 90 90 0)\n");
/* - Wing Cloned to West Position - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*ncell_tr)++;
  ccbw = *ncell_tr;
  strcpy(label,"Wing Cloned to West Position");
  fprintf(lu8,"c     %s\n",label);
  fprintf(lu8,"%5i",*ncell_tr);
  fprintf(lu8," like %i but",cbw);
  fprintf(lu8
   ," *trcl=( 0 0 0 180 90 90 -90 180 90 90 90 0)\n");
  fprintf(lu8,"      u= %i\n",ublade);
  sprintf(buffer," %5i",*ncell_tr);
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"%s\n",label);
  lines(2);
  fprintf(nout,buffer);
  fprintf(nout,"      Cloning Directive:");
  fprintf(nout," like %i but",cbw);
  fprintf(nout
   ," *trcl=( 0 0 0 180 90 90 -90 180 90 90 90 0)\n");
/* - Wing Cloned to South Position - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*ncell_tr)++;
  ccbs = *ncell_tr;
  strcpy(label,"Wing Cloned to South Position");
  fprintf(lu8,"c     %s\n",label);
  fprintf(lu8,"%5i",*ncell_tr);
  fprintf(lu8," like %i but",cbw);
  fprintf(lu8
   ," *trcl=( 0 0 0 -90 180 90 0 -90 90 90 90 0)\n");
  fprintf(lu8,"      u= %i\n",ublade);
  sprintf(buffer," %5i",*ncell_tr);
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"%s\n",label);
  lines(2);
  fprintf(nout,buffer);
  fprintf(nout,"      Cloning Directive:");
  fprintf(nout," like %i but",cbw);
  fprintf(nout
   ," *trcl=( 0 0 0 -90 180 90 0 -90 90 90 90 0)\n");
/* - Region Outside of Blade - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
```

```
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," build_control_blade --\n");
    fprintf(nout," Material Not Found in Linked List label =");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Region Outside of Blade");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&ublade);
/* - Surfaces */
  fprintf(lu8," #%i #%i #%i #%i #%i",ctr,cbw,ccbn,ccbw,ccbs);
  fprintf(lu8,"\n      imp:n=1.0 u= %i\n",ublade);
  sprintf(buffer,"           #%i #%i #%i #%i #%i",ctr,cbw,ccbn,ccbw
   ,ccbs);
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"Definition formed by Complement Function\n");
  lines(1);
  fprintf(nout,buffer);
  *ucb = ublade;
}
```

Function build_control_cells

```
#include <stdio.h>
#include <string.h>
#include <math.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                       } usage_list;

typedef struct su_list{
                         struct su_list *last;
```

```
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;

void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *
 ,char[],surface_usage_list *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_list(char[],int,char[]
 ,char[],char[],surface_usage_list *);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[]
 ,char[],char[],surface_usage_list *);
void bufferpad(char[],int,int);
ll_material *material_match(a_record *,char[],float *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
int *memory_integer(int,int,int *);
void abort();
void lines(int);
int mchar(int *,char[]);

void build_control_cells(int *ncell,int *nuniverse,int ncolcc
 ,int nrowcc,int *ccmap,int *ccmapw,int ncc,FILE *lu8,FILE *lu9
 ,float blade_window_offset,surface_usage_list *ptr_surface_usage
 ,float apitch,int ucb,float cbspan,float cb_stroke,char migt[]
 ,usage_list *ptr_material_usage,a_record *ptr_core_mtls
 ,int *nmaterial,int *ptr_ufa,FILE *lu10,float bypass_density
 ,float dtid,float dtod,int nu_cc,int *fau_fill,int *nsurface){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  b u i l d _ c o n t r o l _ c e l l s  *  creates universes for
 - -                                              unique control cells
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      ncell - Number of Cells in MCNP Model                  (input)
 - -  nuniverse - Number of "Universes" in MCNP Model            (input)
 - -     ncolcc - Number of Columns in Control Cell Map          (input)
 - -     nrowcc - Number of Rows in Control Cell Map             (input)
 - -      ccmap - Map of Control Cell Universe Numbers           (input)
 - -     ccmapw - Map Containing Contents of each Control Cell   (input)
 - -        ncc - Number of Unique Control Cells                 (input)
 - -        lu8 - Stream Pointer for Scratch File for Cell Def-  (input)
 - -              initions
 - -        lu9 - Stream Pointer for Scratch File for Surface    (input)
 - -              Definitions
 - - blade_window_offset
 - -            - sizing of windows for control blades and fuel  (input)
 - -              assemblies in control cell template
 - - ptr_surface_usage
 - -            - linked list for surfaces that have already     (input)
 - -              been defined
```

```
 - -     apitch - assembly pitch                                  (input)
 - -        ucb - control blade universe                          (input)
 - -     cbspan - control blade wing span (cm)                    (input)
 - -  cb_stroke - control blade stroke (cm)                       (input)
 - -       migt - guide tube material                             (input)
 - - ptr_material_usage
 - -            - linked list for materials that have already    (input)
 - -              been defined
 - - ptr_core_mtls
 - -            - linked list for core materials                 (input)
 - -  nmaterial - number of material definitions                  (input)
 - -    ptr_ufa - vector of MNCP "universes" assigned to each     (input)
 - -              fuel assembly
 - -       lu10 - Stream Pointer for Scratch File for Material    (input)
 - -              Definitions
 - - bypass_density
 - -            - density for water in bypass                    (input)
 - -       dtid - inner diameter of guide tube (cm)              (input)
 - -       dtod - outer diameter of guide tube (cm)              (input)
 - -      nu_cc - "universe" index for first control cell        (input)
 - -   fau_fill - array providing fuel assembly universe num-    (output)
 - -              bers filling each unique control cell uni-
 - -              verse
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         i - loop variable for columns in map
 - -         j - loop variable for rows in map
 - -         n - utility loop variable
 - -     index - index for cells and surfaces
 - -   ptr_map - pointer to ccmap array
 - -  ptr_mapw - pointer to ccmapw array
 - -       pcc - pointer to list of unique control cells to track whether
 - -             they have already been defined
 - -        pc - rotating pointer for pcc
 - - n_entries - number of entries in material definition linked list
*/
  short int i,j,n;
  int index, n_entries;
  int *ptr_map = ccmap, *ptr_mapw = ccmapw;
  int *pcc, *pc;
/* - Float Variable(s)
 - -  density - material density
 - -    delta - distance from maximum radial extent of curved corner
 - -            of fuel assembly window to edge of window
 - -  delta_0 - offset for fuel assembly box from control cell
 - -            boundary
 - -        R - radius of curved corner of fuel assembly window
 - -       Xc - x-coordinate of cylinder defining curved corner of
 - -            fuel assembly window
 - -       Yc - Y-coordinate of cylinder defining curved corner of
 - -            fuel assembly window
*/
```

```
  float density, delta, delta_0 = 0.1, R, Xc, Yc;
/* - Character Variable(s)
 - -    label - label for cells and surfaces
 - - material - label for material
 - - mnemonic - MCNP surface mnemonic
 - -    value - surface definition vector
 - -   buffer - string variable for output management
 - -  ptr_buf - pointer to buffer
*/
  char mnemonic[4];
  char *ptr_buf;
  ascii_string label, material, value, buffer;
/* - FILE Variable(s)
 - -  nout - stream pointer to output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to linked list for a specific material
 - -   ptr_ml - pointer to material usage list
*/
  ll_material *ptr_mtl;
  usage_list *ptr_ml;
/* - Fix Up Guide Tube Material Definition for FORTRAN processing - -  */
  { int len = 6, length;
    length = mchar(&len,migt);
    migt[length] = '\0';
  }
/* - Compute coordinates for curved corner of fuel assembly window - - */
  delta = -delta_0+(sqrt(2.0)/2.0)*((dtod/2)+delta_0);
/* - 2.4142 is equivalent to tand(67.5) */
  R = sqrt(2.0)*delta*2.4142;
  Xc = (apitch - delta_0) - R;
  Yc = -(apitch - delta_0) + R;
/* - Set Up Tracking for Control Cell Creation - - - - - - - - - - - - - */
  pcc = memory_integer(1,ncc,pcc);
/* - Sweep Over Map - - - - - - - - - - - - - - - - - - - - - - - - - - -  */
  for(j = 1;j <= ncolcc;j++){
    for(i = 1;i <= nrowcc;i++,ptr_map++){
/* - Check for Null Control Cell */
      if(*ptr_map == 0){
        for(n = 1;n <= 6;n++) ptr_mapw++;
        continue;}
      pc = pcc;
      for(n = 1;n < (*ptr_map-nu_cc+1);n++) pc++;
/* - Check whether the Control Cell that has already been Modeled */
      if(*pc == 1) {
        for(n = 1;n <= 6;n++) ptr_mapw++;
        continue;}
      *pc = 1; (*nuniverse)++;
/* - Build Components of Control Cell and Fill Appropriately - - - - - */
/* - Fuel Assembly in Northwest Quadrant */
/* - Cell */
      sprintf(label,"Fuel Assembly in Northwest Quadrant, CC%i"
       ,*ptr_map);
```

```
      strcpy(material,"Bypass Water");
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," build_control_cells --\n");
        fprintf(nout," Material not found in Usage List, label = %s"
         ,material);
        abort();}
      add_cell(label,material,lu8,ncell,index,-bypass_density
       ,nuniverse);
/* - Surfaces */
      fprintf(lu8," ((");
      sprintf(label,"Window for Fuel Assembly (max X), CC%i"
       ,*ptr_map);
      search_surface_usage_list(1,label,&index,"","","",""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"px");
        sprintf(value,"%10.4E",(apitch-0.1));
        index = -1;
        add_surface(0,label,mnemonic,lu8,lu9,&index,value
         ,ptr_surface_usage,nsurface);}
      else{
        index = -1;
        add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
         ,nsurface);}
      sprintf(label,"Window for Fuel Assembly (min X), CC%i"
       ,*ptr_map);
      search_surface_usage_list(1,label,&index,"","","",""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"px");
        sprintf(value,"%10.4E",(blade_window_offset+0.1));
        add_surface(0,label,mnemonic,lu8,lu9,&index,value
         ,ptr_surface_usage,nsurface);}
      else{
        index = -1;
        add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
         ,nsurface);}
      sprintf(label,"Window for Fuel Assembly (max Y), CC%i"
       ,*ptr_map);
      search_surface_usage_list(1,label,&index,"","","",""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"py");
        sprintf(value,"%10.4E",-(blade_window_offset+0.1));
        index = -1;
        add_surface(0,label,mnemonic,lu8,lu9,&index,value
         ,ptr_surface_usage,nsurface);}
      else{
        index = -1;
        add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
         ,nsurface);}
```

```
sprintf(label,"Window for Fuel Assembly (min Y), CC%i"
 ,*ptr_map);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  strcpy(mnemonic,"py");
  sprintf(value,"%10.4E",-(apitch-0.1));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);}
else{
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);}
fprintf(lu8," )\n      (#(");
sprintf(label,"Window for Fuel Assembly (max X), CC%i",*ptr_map);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," build_control_cells --\n");
  fprintf(nout," Surface not Found in Usage List, label = %s"
   ,label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label,"Window for Fuel Assembly (min Y), CC%i",*ptr_map);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," build_control_cells --\n");
  fprintf(nout," Surface not Found in Usage List, label = %s"
   ,label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label,"Curved Corner in Window for Fuel Assembly, CC%i"
 ,*ptr_map);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  strcpy(mnemonic,"c/z");
  sprintf(value,"%10.4E %10.4E %10.4E",Xc,Yc,R);
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);}
else
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
sprintf(label
 ,"Y Ambiguity Surface for Fuel Assembly Window, CC%i"
 ,*ptr_map);
```

```
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"py");
        sprintf(value,"%10.4E",Yc);
        index = -1;
        add_surface(0,label,mnemonic,lu8,lu9,&index,value
         ,ptr_surface_usage,nsurface);}
      else{
        index = -index;
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);}
      sprintf(label
       ,"X Ambiguity Surface for Fuel Assembly Window, CC%i"
       ,*ptr_map);
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"px");
        sprintf(value,"%10.4E",Xc);
        index = 1;
        add_surface(0,label,mnemonic,lu8,lu9,&index,value
         ,ptr_surface_usage,nsurface);}
      else
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);
      fprintf(lu8," )))");
      fprintf(lu8,"\n      u= %i imp:n= 1.0\n",*ptr_map);
/* - Define Translation */
      { float trans;
        trans = apitch+blade_window_offset;
        fprintf(lu8,"      *trcl=( 0.0 0.0 0.0 "
         ,-trans,trans);
        fprintf(lu8," 180 90 90 -90 180 90 90 90 0 )\n");
        lines(1);
        fprintf(nout,"     *trcl=( %10.4E %10.4E 0.0 "
         ,-trans,trans);
        fprintf(nout," 180 90 90 -90 180 -90 90 90 0 )");
      }
/* -  Determine Lattice with which to Fill Window */
      if(*ptr_mapw != 0){
        { short int nn;
          int *p = ptr_ufa;
/*          for(nn = 1; nn < (*ptr_mapw-nu_cc+1);nn++,p++) */
          for(nn = 1;nn < *ptr_mapw;nn++,p++)
           continue;
          fprintf(lu8,"      fill= %i\n",*p);
          lines(1);
          fprintf(nout,"\n     fill= %i\n",*p);
          *fau_fill = *p;
          fau_fill++;
        }
      }
      else{
```

```
        fprintf(lu8,"      fill= 0\n");
        fprintf(nout," fill= 0\n");
        *fau_fill = 0;
        fau_fill++;}
      ptr_mapw++;
/* - Fuel Assembly in Northeast Quadrant */
/* - Cell */
      sprintf(label,"Fuel Assembly in Northeast Quadrant, CC%i"
       ,*ptr_map);
      fprintf(lu8,"c      %s\n",label);
      (*ncell)++;
      fprintf(lu8,"%5i like %i but",*ncell,((*ncell)-1));
      lines(2);
      sprintf(buffer," %5i",*ncell);
      bufferpad(buffer,strlen(buffer),59);
      ptr_buf = buffer+59*sizeof(char);
      sprintf(ptr_buf,"%s",label);
      fprintf(nout,"%s\n",buffer);
      fprintf(nout
       ,"     Cloning Directive: like %i but *trcl=( 0 0 0"
       ,((*ncell)-1));
      fprintf(nout," 90 0 90 180 90 90 90 90 0)\n");
      if(*ptr_mapw != 0)
/* -  Determine Lattice with which to Fill Window */
        { short int nn;
          int *p = ptr_ufa;
/*          for(nn = 1; nn < (*ptr_mapw-nu_cc+1);nn++,p++) */
          for(nn = 1;nn < *ptr_mapw;nn++,p++)
           continue;
          fprintf(lu8," fill= %i\n",*p);
          lines(1);
          fprintf(nout,"      fill= %i\n",*p);
          *fau_fill = *p;
          fau_fill++;
        }
      else{
        fprintf(lu8," fill= 0\n");
        fprintf(nout," fill= 0\n");
        *fau_fill = 0;
        fau_fill++;}
      fprintf(lu8
       ,"      *trcl=( 0 0 0 90 0 90 180 90 90 90 90 0)\n");
      fprintf(lu8,"      u= %i imp:n= 1.0\n",*ptr_map);
      ptr_mapw++;
/* - Fuel Assembly in Southeast Quadrant */
/* - Cell */
      sprintf(label,"Fuel Assembly in Southeast Quadrant, CC%i"
       ,*ptr_map);
      fprintf(lu8,"c     %s\n",label);
      (*ncell)++;
      fprintf(lu8,"%5i like %i but",*ncell,((*ncell)-1));
      lines(2);
      sprintf(buffer," %5i",*ncell);
      bufferpad(buffer,strlen(buffer),59);
```

```
      ptr_buf = buffer+59*sizeof(char);
      sprintf(ptr_buf,"%s",label);
      fprintf(nout,"%s\n",buffer);
      fprintf(nout
       ,"     Cloning Directive: like %i but trcl=( 0 0 0)\n"
       ,((*ncell)-1));
      if(*ptr_mapw != 0)
/* -  Determine Lattice with which to Fill Window */
        { short int nn;
          int *p = ptr_ufa;
/*          for(nn = 1; nn < (*ptr_mapw-nu_cc+1);nn++,p++)  */
          for(nn = 1;nn < *ptr_mapw;nn++,p++)
           continue;
          fprintf(lu8," fill= %i\n",*p);
          lines(1);
          fprintf(nout,"      fill= %i\n",*p);
          *fau_fill = *p;
          fau_fill++;
        }
      else{
        fprintf(lu8," fill= 0\n");
        fprintf(nout," fill= 0\n");
        *fau_fill = 0;
        fau_fill++;}
      fprintf(lu8
       ,"      trcl=( 0 0 0 )");
      fprintf(lu8," u= %i imp:n= 1.0\n",*ptr_map);
      ptr_mapw++;
/* - Fuel Assembly in Southwest Quadrant */
/* - Cell */
      sprintf(label,"Fuel Assembly in Southwest Quadrant, CC%i"
       ,*ptr_map);
      fprintf(lu8,"c      %s\n",label);
      (*ncell)++;
      fprintf(lu8,"%5i like %i but",*ncell,((*ncell)-3));
      lines(2);
      sprintf(buffer," %5i",*ncell);
      bufferpad(buffer,strlen(buffer),59);
      ptr_buf = buffer+59*sizeof(char);
      sprintf(ptr_buf,"%s",label);
      fprintf(nout,"%s\n",buffer);
      fprintf(nout
       ,"     Cloning Directive: like %i but *trcl=( 0 0 0"
       ,((*ncell)-2));
      fprintf(nout," -90 180 90 0 -90 90 90 90 0)\n");
      if(*ptr_mapw != 0){
/* -  Determine Lattice with which to Fill Window */
        { short int nn;
          int *p = ptr_ufa;
/*          for(nn = 1; nn < (*ptr_mapw-nu_cc+1);nn++,p++) */
          for(nn = 1;nn < *ptr_mapw;nn++,p++)
           continue;
          fprintf(lu8," fill= %i\n",*p);
          lines(1);
```

```
          fprintf(nout," fill= %i\n",*p);
          *fau_fill = *p;
          fau_fill++;
        }
      }
      else{
        fprintf(lu8," fill= 0\n");
        fprintf(nout," fill= 0\n");
        *fau_fill = 0;
        fau_fill++;}
      fprintf(lu8
       ,"      *trcl=( 0 0 0 -90 180 90 0 -90 90 90 90 0)");
      fprintf(lu8," u= %i imp:n= 1.0\n",*ptr_map);
      ptr_mapw++;
/* - Control Blade Window */
/* - Cell */
      sprintf(label,"Blade Window, CC%i",*ptr_map);
      strcpy(material,"Bypass Water");
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," build_control_cells -- \n");
        fprintf(nout," Material not found in Usage List, label = %s"
         ,material);
        abort();}
     add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Horizontal Part of Cross */
      fprintf(lu8," (");
      strcpy(label,"Surface 1 for Control Blade Window");
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"py");
        sprintf(value,"%10.4E",-blade_window_offset);
        index = 1;
        add_surface(0,label,mnemonic,lu8,lu9,&index
         ,value,ptr_surface_usage,nsurface);}
      else
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);
      strcpy(label,"Surface 2 for Control Blade Window");
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"py");
        sprintf(value,"%10.4E",blade_window_offset);
        index = -1;
        add_surface(0,label,mnemonic,lu8,lu9,&index
         ,value,ptr_surface_usage,nsurface);}
      else{
        index = -index;
        add_surface(1,label,"",lu8,lu9,&index,""
```

```
        ,ptr_surface_usage,nsurface);}
      strcpy(label,"Surface 3 for Control Blade Window");
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"px");
        sprintf(value,"%10.4E",-(cbspan+0.1));
        index = 1;
        add_surface(0,label,mnemonic,lu8,lu9,&index
         ,value,ptr_surface_usage,nsurface);}
      else
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);
      strcpy(label,"Surface 4 for Control Blade Window");
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"px");
        sprintf(value,"%10.4E",(cbspan+0.1));
        index = -1;
        add_surface(0,label,mnemonic,lu8,lu9,&index
         ,value,ptr_surface_usage,nsurface);}
      else{
        index = -index;
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);}
      fprintf(lu8," ):(");
/* - Vertical Part of Cross */
      strcpy(label,"Surface 5 for Control Blade Window");
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"px");
        sprintf(value,"%10.4E",-blade_window_offset);
        index = 1;
        add_surface(0,label,mnemonic,lu8,lu9,&index
         ,value,ptr_surface_usage,nsurface);}
      else
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);
      strcpy(label,"Surface 6 for Control Blade Window");
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"px");
        sprintf(value,"%10.4E",blade_window_offset);
        index = -1;
        add_surface(0,label,mnemonic,lu8,lu9,&index
         ,value,ptr_surface_usage,nsurface);}
      else{
        index = -index;
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);}
      strcpy(label,"Surface 7 for Control Blade Window");
```

```
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"py");
        sprintf(value,"%10.4E",-(cbspan+0.1));
        index = 1;
        add_surface(0,label,mnemonic,lu8,lu9,&index
         ,value,ptr_surface_usage,nsurface);}
      else
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);
      strcpy(label,"Surface 8 for Control Blade Window");
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        strcpy(mnemonic,"py");
        sprintf(value,"%10.4E",(cbspan+0.1));
        index = -1;
        add_surface(0,label,mnemonic,lu8,lu9,&index
         ,value,ptr_surface_usage,nsurface);}
      else{
        index = -index;
        add_surface(1,label,"",lu8,lu9,&index,""
         ,ptr_surface_usage,nsurface);}
      fprintf(lu8,")\n");
      fprintf(lu8,"      u= %i imp:n= 1.0\n",*ptr_map);
      if(*ptr_mapw >= 0){
        { int notch_fi=48;
          float trans;
          trans =
           -(*ptr_mapw)*(cb_stroke/((float) notch_fi));
          fprintf(lu8
           ,"      fill= %i trcl=( 0.0 0.0 %10.4E )\n"
           ,ucb,trans);
          lines(1);
          fprintf(nout,"     fill= %i trcl= ( 0.0 0.0 %10.4E)\n"
           ,ucb,trans);
        }
      }
      ptr_mapw++;
/* - Guide Tube */
      if(*ptr_mapw != 0){
        { short int ng, ngt;
          int val;
          if(*ptr_mapw > 9)
            ngt = 2;
          else
            ngt = 1;
          for(ng = 1;ng <= ngt;ng++){
            if(ngt == 1) val = *ptr_mapw;
            else
              if(ng == 1) val = (*ptr_mapw)/10;
              else val = (*ptr_mapw)-(10*val);
/* - Cell */
```

```
            if(ng == 1)
              sprintf(label,"Guide Tube Segment, CC%i",*ptr_map);
            else
              sprintf(label,"Guide Tube Segment (2), CC%i",*ptr_map);
            strcpy(material,migt);
            search_usage_list(1,migt,&index,ptr_material_usage);
            if(index == 0){
              ptr_mtl = material_match(ptr_core_mtls,migt,&density
               ,&n_entries);
              (*nmaterial)++;
              ptr_ml = ptr_material_usage;
              while(ptr_ml->next != NULL)  ptr_ml = ptr_ml->next;
              index = (ptr_ml->index)+1;
              ptr_ml = load_usage_list(migt,index,ptr_ml);}
            else{
              (void) material_match(ptr_core_mtls,migt,&density
               ,&n_entries);
              n_entries = 0;}
            add_cell(label,migt,lu8,ncell,index,-density,nuniverse);
/* - Surfaces */
            strcpy(label,"Outer Surface of Guide Tube");
            search_surface_usage_list(1,label,&index,"","","""
             ,ptr_surface_usage);
            if(index == 0){
              strcpy(mnemonic,"c/z");
              sprintf(value,"%10.4E %10.4E %10.4E",apitch,-apitch
               ,(dtod/2.0));
              index = -1;
              add_surface(0,label,mnemonic,lu8,lu9,&index,value
               ,ptr_surface_usage,nsurface);}
            else{
              index = -index;
              add_surface(1,label,"",lu8,lu9,&index,""
               ,ptr_surface_usage,nsurface);}
            strcpy(label,"Inner Surface of Guide Tube");
            search_surface_usage_list(1,label,&index,"","","""
             ,ptr_surface_usage);
            if(index == 0){
              strcpy(mnemonic,"c/z");
              sprintf(value,"%10.4E %10.4E %10.4E",apitch,-apitch
               ,(dtid/2));
              index = 1;
              add_surface(0,label,mnemonic,lu8,lu9,&index,value
               ,ptr_surface_usage,nsurface);}
            else
              add_surface(1,label,"",lu8,lu9,&index,""
               ,ptr_surface_usage,nsurface);
/* - Material */
            if(n_entries != 0){
              add_material(lu10,nmaterial,migt,n_entries,ptr_mtl);
              n_entries = 0;}
            switch (val) {
              case 1:
                fprintf(lu8,"\n      trcl=( %10.4E %10.4E 0.0)"
```

```
                ,-(2*apitch),(2*apitch));
              lines(1);
              fprintf(nout,"     trcl=( %10.4E %10.4E 0.0)\n"
               ,-(2*apitch),(2*apitch));
              break;
            case 2:
              fprintf(lu8,"\n      trcl=( 0.0 %10.4E 0.0)"
               ,(2*apitch));
              lines(1);
              fprintf(nout,"     trcl=( 0.0 %10.4E 0.0)\n"
               ,(2*apitch));
              break;
            case 3:
              break;
            case 4:
              fprintf(lu8,"\n      trcl=( %10.4E 0.0 0.0)"
               ,-(2*apitch));
              lines(1);
              fprintf(nout,"     trcl=( %10.4E 0.0 0.0)\n"
               ,-(2*apitch));
          }
          fprintf(lu8," u= %i imp:n= 1.0\n",*ptr_map);
/* - Inside of Guide Tube */
/* - Cell */
          if(ng == 1)
            sprintf(label,"Inside Guide Tube Segment, CC%i"
             ,*ptr_map);
          else
            sprintf(label,"Inside Guide Tube Segment (2), CC%i"
             ,*ptr_map);
          strcpy(material,"void");
          add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
/* - Surfaces */
          strcpy(label,"Inner Surface of Guide Tube");
          search_surface_usage_list(1,label,&index,"","","""
           ,ptr_surface_usage);
          if(index == 0){
            lines(3);
            fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
            fprintf(nout," build_control_cells --\n");
            fprintf(nout
             ,"0Surface not Found in Usage List, label = %s",label);
            abort();}
          index = -index;
          add_surface(1,label,"",lu8,lu9,&index,""
           ,ptr_surface_usage,nsurface);
          switch (val) {
            case 1:
              fprintf(lu8,"      trcl=( %10.4E %10.4E 0.0)"
               ,-(2*apitch),(2*apitch));
              lines(1);
              fprintf(nout,"     trcl=( %10.4E %10.4E 0.0)\n"
               ,-(2*apitch),(2*apitch));
              break;
```

```
              case 2:
                fprintf(lu8,"      trcl=( 0.0 %10.4E 0.0)"
                 ,(2*apitch));
                lines(1);
                fprintf(nout,"     trcl=( 0.0 %10.4E 0.0)\n"
                 ,(2*apitch));
                break;
              case 3:
                break;
              case 4:
                fprintf(lu8,"      trcl=( %10.4E 0.0 0.0)"
                 ,-(2*apitch));
                lines(1);
                fprintf(nout,"     trcl=( %10.4E 0.0 0.0)\n"
                 ,-(2*apitch));
            }
            fprintf(lu8," u= %i imp:n= 1.0\n",*ptr_map);
          }
        }
      }
/* - Balance of Universe */
/* - Cell */
      sprintf(label,"Balance of Control Cell, CC%i",*ptr_map);
      strcpy(material,"Bypass Water");
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
        fprintf(nout
         ,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," build_control_cells --\n");
        fprintf(nout
         ," Material not Found in Usage List, label = %s\n"
         ,material);
        abort();
      }
      (*ncell)++;
      fprintf(lu8,"c     %s\n",label);
      sprintf(buffer," %5i",*ncell);
      bufferpad(buffer,strlen(buffer),49);
      ptr_buf = buffer+49*sizeof(char);
      sprintf(ptr_buf,"%4i",*nuniverse);
      bufferpad(buffer,strlen(buffer),59);
      ptr_buf = buffer+59*sizeof(char);
      sprintf(ptr_buf,"%s",label);
      lines(1);
      fprintf(nout,"%s\n",buffer);
/* - Defintion by Complement */
     { short int nn;
       int nloc;
       if(*ptr_mapw == 0)
         nloc = (*ncell)-5;
       else
         if(*ptr_mapw < 10)
           nloc = (*ncell)-7;
```

```
          else
            nloc = (*ncell)-9;
        fprintf(lu8,"%5i",*ncell);
        fprintf(lu8," %i %10.4E",index,-bypass_density);
        lines(1);
        fprintf(nout,"      ");
        for(nn = nloc;nn < *ncell;nn++){
          fprintf(lu8," #%i",nn);
          fprintf(nout," #%i",nn);}
      }
      fprintf(nout,"\n");
      fprintf(lu8,"\n      u= %i imp:n= 1.0\n",*ptr_map);
/* - End of Control Cell Construction - - - - - - - - - - - - - */
       ptr_mapw++;
     }
   }
   pcc = memory_integer(-1,ncc,pcc);
   return;
}
```

Subroutine ccmgen

```
      SUBROUTINE CCMGEN(NROWP,NCOLP,MMAP,NROWCC,NCOLCC,CCMAP,CCMAPW
     2 ,CORE_F,NCOLBP,NROWBP,BLADEP,NCC,ICLOC,NROWB,NCOLB)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  CCMGEN *  Creates Map of Unique Control Cells
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -       NROWP - Maximum Number of Rows of Fuel Assemblies   (input)
C - -               in Core
C - -       NCOLP - Maximum Number of Columns of Fuel Assem-    (input)
C - -               blies in Core
C - -        MMAP - Map of Fuel Assembly Material Types in the  (input)
C - -               Core
C - -      NROWCC - Number of Rows in Control Cell Map          (input)
C - -      NCOLCC - Number of Columns in Control Cell Map       (input)
C - -       CCMAP - Map of Unique Control Cell Types           (output)
C - -      CCMAPW - Scratch Array for Identifying Each Con-   (scratch)
C - -               trol Cell in Terms of Constituent Fuel
C - -               Assemblies and Control Blade Position
C - -               (1 - Fuel Assembly Type in the Northwest
C - -                    Quadrant
C - -                2 - Fuel Assembly Type in the Northeast
C - -                      Quadrant
C - -                3 - Fuel Assembly Type in the Southeast
C - -                    Quadrant
C - -                4 - Fuel Assembly Type in the Southwest
C - -                    Quadrant
C - -                5 - Blade Position
C - -                6 - Location of Incore Detector in Control
C - -                    Cell
C - -                    0 - No Guide Tube Present,
C - -                    1 - Guide Tube in Northwest Quadrant
C - -                    2 - Guide Tube in Northeast Quadrant
C - -                    3 - Guide Tube in Southeast Quadrant
C - -                    4 - Guide Tube in Southwest Quadrant
C - -                    If there is a second Guide Tube in
C - -                    Control Cell, then 10 Times the
C - -                    Index above is Added to the Value)
C - -      CORE_F - Fraction of Core Represented                (input)
C - -               (0 - Full Core,
C - -                2 - Half Core,
C - -                4 - Quarter Core)
C - -     NCOLBP - Number of Columns in Control Blade Map       (input)
C - -     NROWBP - Number of Rows in Control Blade Map          (input)
C - -     BLADEP - Control Blade Insertion Map                  (input)
C - -        NCC - Number of MCNP "Universes"                     (i&o)
C - -      INLOC - Locations of Incore Detectors                (input)
C - -      NROWB - Number of Rows of Control Blade Locations in (input)
C - -              the Whole Core
C - -      NCOLB - Number of Columns of Control Blade Locations (input)
C - -              in the Whole Core
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
```

```
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  EVENX - Flag to Indicate that Peripheral Fuel Assemblies not
C - -          Resident in Control Cells Exist at the X extreme(s) of
C - -          the Map
C - -  EVENY - Flag to Indicate that Peripheral Fuel Assemblies not
C - -          Resident in Control Cells Exist at the Y extreme of
C - -          the Map
C - -   NULL - Flag to Indicate that a Control Cell Contents Vector
C - -          Empty, Indicating a Null Cell
C - -  MATCH - Flag to Indicate that a Given Control Cell matches a
C - -          Reference Control Cell
      LOGICAL EVENX,EVENY,NULL,MATCH
      INTEGER CORE_F
      INTEGER CCMAP(NCOLCC,NROWCC),BLADEP(NCOLBP,NROWBP)
      INTEGER CCMAPW(6,NCOLCC,NROWCC)
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MMAP(NCOLP,NROWP),ICLOC(NCOLB,NROWB)
C - - Determine Whether there are Peripheral Assemlies not in Control
C - - Cells at the Extremes of the Material Map
      IF(CORE_F .EQ. 4) THEN
        IF(FLOAT(NROWP/2) .EQ. (FLOAT(NROWP)/2.)) THEN
          EVENX = .TRUE.
          EVENY = .TRUE.
        ELSE
          EVENX = .FALSE.
          EVENY = .FALSE.
        ENDIF
      ELSEIF(CORE_F .EQ. 1) THEN
        IF(FLOAT(NROWP/4) .EQ. (FLOAT(NROWP)/4.)) THEN
          EVENX = .TRUE.
          EVENY = .TRUE.
        ELSE
          EVENX = .FALSE.
          EVENY = .FALSE.
        ENDIF
      ELSEIF(CORE_F .EQ. 2) THEN
        IF(FLOAT(NROWP/4) .EQ. (FLOAT(NROWP)/4.)) THEN
          EVENX = .TRUE.
        ELSE
          EVENX = .FALSE.
        ENDIF
        IF(FLOAT(NCOLP/2) .EQ. (FLOAT(NCOLP)/2.)) THEN
          EVENY = .TRUE.
        ELSE
          EVENY = .FALSE.
        ENDIF
      ENDIF
C - - Sweep through Map Loading Identifications for Control
C - - Cells
      IXCC = 0
      IYCC = 1
      CALL INVALI((NCOLCC*NROWCC),CCMAP,0)
      CALL INVALI((6*NCOLCC*NROWCC),CCMAPW,0)
C - - Set up Starting and Ending Limits for Processing Map
```

```
      IF(EVENY) THEN
        JSTART = 2
        JEND = NROWP-1
      ELSE
        JSTART = 1
        JEND = NROWP
      ENDIF
      IF(EVENX) THEN
        ISTART = 2
        IEND = NCOLP-1
      ELSE
        ISTART = 1
        IEND = NCOLP
      ENDIF
      IF(CORE_F .EQ. 4) THEN
        IEND = NCOLP-1
        JEND = NROWP-1
      ENDIF
      IF(CORE_F .EQ. 2) JEND = NROWP-1
C - - Load Locations for Extra Row at the Top of the Problem
      IYCC = 1
      IF(EVENY) THEN
        DO I = ISTART,IEND,2
         IXCC = IXCC+1
         CCMAPW(4,IXCC,IYCC) = MMAP(I,1)
         CCMAPW(3,IXCC,IYCC) = MMAP((I+1),1)
        ENDDO
        IF(CORE_F .EQ. 4) THEN
          IXCC = IXCC+1
          CCMAPW(4,IXCC,IYCC) = MMAP(NROWP,1)
        ENDIF
      ENDIF
C - - Process Bulk of Map
      DO J = JSTART,JEND,2
        IXCC = 0
C - - Process First Column if Extra Column
        IF(EVENX) THEN
          IXCC = IXCC+1
          CCMAPW(2,IXCC,IYCC) = MMAP(1,J)
          CCMAPW(3,IXCC,IYCC) = MMAP(1,(J+1))
        ENDIF
C - - Process Bulk of Map
        DO I = ISTART,IEND,2
          IXCC = IXCC+1
          CCMAPW(1,IXCC,IYCC) = MMAP(I,J)
          CCMAPW(2,IXCC,IYCC) = MMAP((I+1),J)
          CCMAPW(4,IXCC,IYCC) = MMAP(I,(J+1))
          CCMAPW(3,IXCC,IYCC) = MMAP((I+1),(J+1))
          CCMAPW(5,IXCC,IYCC) = BLADEP(IXCC,IYCC)
          IF(ICLOC(IXCC,IYCC) .EQ. 1) THEN
            IF(CCMAPW(6,IXCC,IYCC) .EQ. 0) THEN
              CCMAPW(6,IXCC,IYCC) = 3
            ELSE
              CCMAPW(6,IXCC,IYCC) = CCMAPW(6,IXCC,IYCC)+30
```

```
            ENDIF
            IF(CCMAPW(6,(IXCC+1),IYCC) .EQ. 0) THEN
              CCMAPW(6,(IXCC+1),IYCC) = 4
            ELSE
              CCMAPW(6,(IXCC+1),IYCC) = CCMAPW(6,(IXCC+1),IYCC)+40
            ENDIF
            IF(CCMAPW(6,(IXCC+1),(IYCC+1)) .EQ. 0) THEN
              CCMAPW(6,(IXCC+1),(IYCC+1)) = 1
            ELSE
              CCMAPW(6,(IXCC+1),(IYCC+1)) =
     2         CCMAPW(6,(IXCC+1),(IYCC+1))+10
            ENDIF
            IF(CCMAPW(6,IXCC,(IYCC+1)) .EQ. 0) THEN
              CCMAPW(6,IXCC,(IYCC+1)) = 2
            ELSE
              CCMAPW(6,IXCC,(IYCC+1)) = CCMAPW(6,IXCC,(IYCC+1))+20
            ENDIF
          ENDIF
        ENDDO
C - - Process Last Column, if Extra Column or Quarter Core
        IF(EVENX .OR. (CORE_F .EQ. 4)) THEN
          IXCC = IXCC+1
          CCMAPW(1,IXCC,IYCC) = MMAP(NCOLP,J)
          CCMAPW(4,IXCC,IYCC) = MMAP(NCOLP,(J+1))
          CCMAPW(5,IXCC,IYCC) = BLADEP(IXCC,IYCC)
        ENDIF
        IYCC = IYCC+1
      ENDDO
C - - Load Locations for Extra Row at Bottom of Problem, if Present
C - - or Quarter or Half Core
      IF(EVENY .OR. (CORE_F .EQ. 2) .OR. (CORE_F .EQ. 4)) THEN
        IXCC = 0
        DO I = ISTART,IEND,2
          IXCC = IXCC+1
          CCMAPW(1,IXCC,IYCC) = MMAP(I,NROWP)
          CCMAPW(2,IXCC,IYCC) = MMAP((I+1),NROWP)
          CCMAPW(5,IXCC,IYCC) = BLADEP(IXCC,IYCC)
        ENDDO
        IF(CORE_F .EQ. 4) THEN
          IXCC = IXCC+1
          CCMAPW(1,IXCC,IYCC) = MMAP(NCOLP,NROWP)
          CCMAPW(5,IXCC,IYCC) = BLADEP(IXCC,IYCC)
        ENDIF
      ENDIF
C - - Search for Unique Control Cells - - - - - - - - - - - - - - - - -
      CALL INVALI((NCOLCC*NROWCC),CCMAP,0)
      DO J = 1,NROWCC
        DO I = 1,NCOLCC
C - - Check for non-fuel Cells
          NULL = .TRUE.
          DO N = 1,4
            IF(CCMAPW(N,I,J) .NE. 0) NULL = .FALSE.
          ENDDO
          IF(.NOT. NULL) THEN
```

```
C - - Match Found
            IF(CCMAP(I,J) .EQ. 0) THEN
              NCC = NCC+1
              CCMAP(I,J) = NCC
C - - Examine Balance of Core for Matches with Current Control Cell
              IF(I .NE. NCOLCC) THEN
                DO II = (I+1),NCOLCC
                  MATCH = .TRUE.
                  DO N = 1,6
                    IF(CCMAPW(N,II,J) .NE. CCMAPW(N,I,J))
     2                MATCH = .FALSE.
                  ENDDO
                  IF(MATCH) CCMAP(II,J) = NCC
                ENDDO
              ENDIF
              IF(J .NE. NROWCC) THEN
                DO JJ = (J+1),NROWCC
                  DO II = 1,NCOLCC
                    MATCH = .TRUE.
                    DO N = 1,6
                      IF(CCMAPW(N,II,JJ) .NE. CCMAPW(N,I,J))
     2                  MATCH = .FALSE.
                    ENDDO
                    IF(MATCH) CCMAP(II,JJ) = NCC
                  ENDDO
                ENDDO
              ENDIF
            ENDIF
          ENDIF
        ENDDO
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Function core_lattice_generation

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                            struct ascii_record *last;
                            ascii_string line;
                            struct ascii_record *next;
                           } a_record;
typedef struct s_material{
                          struct s_material *last;
                          int atomic_number;
                          int mass_number;
                          float weight_percentage;
                          char library_suffix[5];
                          struct s_material *next;
                         } ll_material;
```

```
typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;


typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                       } surface_usage_list;

void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_symmetry_surfaces(int,FILE *,FILE *,surface_usage_list *
 ,int);

void core_lattice_generation(int *ncell,int *nsurface,float apitch
 ,float bypass_density,surface_usage_list *ptr_surface_usage
 ,usage_list *ptr_material_usage,int *ccmap,int ncolcc,int nrowcc
 ,FILE *lu8,FILE *lu9,int core_f,int *nuniverse,float afl){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  c o r e _ l a t t i c e _ g e n e r a t i o n  *
 - -   Creates Control Cell Lattice Field
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      ncell - Total Number of Cells in MCNP Model              (i|o)
 - -     apitch - Fuel Assembly Pitch (cm)                        (input)
 - - bypass_density
 - -            - water density in bypass region (g/cc)           (input)
 - -  ptr_surface_usage
 - -            - pointer to linked list for surface labels and   (input)
 - -              indices
 - -  ptr_material_usage
 - -            - pointer to linked list for material labels and (input)
 - -              indices
 - -      ccmap - pointer to control cell map                      (input)
 - -     ncolcc - number of columns in control cell map            (input)
 - -     nrowcc - number of rows in control cell map               (input)
 - -        lu8 - pointer to scratch file containing MCNP cell    (input)
 - -              definitions
 - -        lu9 - pointer to scratch file containing MCNP sur-    (input)
 - -              face definitions
 - -     core_f - flag for fraction of core modeled                (input)
```

```
 - -              (1 - full, 2 - half, 4 - quarter)
 - -  nuniverse - number of "universes" in the MCNP deck          (input)
 - -        afl - active fuel length (cm)                          (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  max_n_span - maximum span of nodal members from origin
 - -       index - index for referencing lists
 - -        zero - the value of zero
*/
  int max_n_span = 0, index, zero = 0;
/* - Character Variable(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - -   buffer - string variable to hold output fields
 - -    label - string variable to hold variables labels for editing
 - - material - string variable to hold name of cell material
 - - mnemonic - MCNP identifier for surface
 - -    value - string variable to hold value of surface defintion
 - -  ptr_buf - pointer to buffer string variable
*/
  char mnemonic[4];
  ascii_string buffer, label, value, material;
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Create Cell for Active Core - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  strcpy(material,"Bypass Water");
  strcpy(label,"Active Core");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function ");
    fprintf(nout," core_lattice_generation --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,&zero);
/* - Surfaces */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function ");
    fprintf(nout," core_lattice_generation --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,label);
    abort();}
  strcpy(value,"0.0");
  strcpy(mnemonic,"");
```

```
  index = -index;
  add_surface(1,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Active Fuel */
  strcpy(label,"Top of Active Fuel");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function ");
    fprintf(nout," core_lattice_generation --\n");
    fprintf(nout," Surface Not Found in Linked List,label = %s\n"
     ,label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Active Fuel */
  strcpy(label,"Bottom of Active Fuel");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function ");
    fprintf(nout," core_lattice_generation --\n");
    fprintf(nout," Surface Not Found in Linked List, surface = %s\n"
     ,label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Filling Universe and Neutron Importance of Cell */
  (*nuniverse)++;
  fprintf(lu8,"\n      fill=%i\n",*nuniverse);
  fprintf(lu8,"      imp:n= 1.0\n");
/* - Create Control Cell Lattice - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  strcpy(label,"Core Lattice");
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function ");
    fprintf(nout," core_lattice_generation --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
  strcpy(label,"Edge of Control Cell (max X)");
  sprintf(value,"%10.4E",apitch);
  index = -1;
```

```
  add_surface(0,label,"px",lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"Edge of Control Cell (min X)");
  sprintf(value,"%10.4E",(-apitch));
  index = 1;
  add_surface(0,label,"px",lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"Edge of Control Cell (min Y)");
  sprintf(value,"%10.4E",(-apitch));
  index = 1;
  add_surface(0,label,"py",lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  strcpy(label,"Edge of Control Cell (max Y)");
  sprintf(value,"%10.4E",apitch);
  index = -1;
  add_surface(0,label,"py",lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n");
  fprintf(lu8,"      lat=1 u=%i imp:n= 1.0\n",*nuniverse);
/* - Populate Lattice */
  fprintf(lu8,"      fill=");
  if(core_f == 4)
    fprintf(lu8," %i:0 %i:0 0:0\n",-ncolcc,-nrowcc);
  if(core_f == 1)
    if(ncolcc%2 != 0)
      fprintf(lu8," %i:%i %i:%i 0:0\n"
       ,(-(ncolcc/2)),(ncolcc/2),(-(nrowcc/2)),(nrowcc/2));
    else
      fprintf(lu8," %i:%i %i:%i 0:0\n"
       ,(-((ncolcc/2)+1)),((ncolcc/2)+1)
       ,(-((nrowcc/2)+1)),((nrowcc/2)+1));
  if(core_f == 2){
    if(ncolcc%2 != 0)
      fprintf(lu8," %i:%i",(-(ncolcc/2)),(ncolcc/2));
    else
      fprintf(lu8," %i:%i"
       ,(-((ncolcc/2)+1)),((ncolcc/2)+1));
    fprintf(lu8," %i:0 0:0 \n",(-(ncolcc/2)));}
  { short int i,j;
    int *loc;
    loc = ccmap;
    fprintf(lu8,"      ");
    for(i = -1;i < ncolcc;i++)
     fprintf(lu8," %3i",*nuniverse);
    fprintf(lu8,"\n");
    for(j = 0;j < nrowcc;j++){
      fprintf(lu8,"      ");
      fprintf(lu8," %3i",*nuniverse);
      for(i = 0;i < ncolcc;i++){
        if(*loc != 0)
          fprintf(lu8," %3i",*loc);
        else
          fprintf(lu8," %3i",*nuniverse);
        loc++;}
```

```
        fprintf(lu8,"\n");
      }
    }
  }
```

Function echo_MCNP_deck

```
#include<stdio.h>
#include<string.h>

void echo_MCNP_deck(FILE *lmcnp){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -    * e c h o _ M C N P _ d e c k *  Copies the MNCP File to the
 - -                                      Output Stream
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -    Argument(s):
 - -         lmcnp - File Pointer to MCNP Input Deck             (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declaration(s) - - - - - - - - - - - - - - - - - - - - - -
/* - Character Variable(s)
 - -           c - single character used to read from input file
 - -           p - pointer to character variable
 - -      buffer - string variable used to copy files from input file
 - - case_title - title for analysis
*/
  unsigned char c;
  char *p;
  char buffer[133] = "";
/* - FILE Variable(s)
 - -   nout - output file
*/
  extern FILE *nout;
/* - - Rewind File before Processing - - - - - - - - - - - - - - - - - */
  rewind(lmcnp);
  lines(4);
  fprintf(nout,"0Input to MCNP\n");
  fprintf(nout,"\n");
/* - Copy Line-by-line to Output File - - - - - - - - - - - - - - - - -*/
  do
  { strcpy(buffer,"");
    p = buffer;
    do
    { c = fgetc(lmcnp);
      *p = c;
      p++;}
    while((c != 10) && (c != 255));
    if(c != 255) fprintf(nout," ");
    if(c != 255){
      p = buffer;
      c = *p;
      while(c != 10){
        fputc(*p,nout);
        p++;
        c = *p;
      }
      fprintf(nout,"\n");
      lines(1);
    }
```

```
  }
  while(c != 255);
  fclose(lmcnp);
  return;
}
```

Function edit_materials

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                       } usage_list;

void header();
void lines(int);

void edit_materials(usage_list *ptr_material_usage){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * e d i t _ m a t e r i a l s  *  edits the descriptions of mater-
 - -                                   ials used in the problem
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -  material_usage_list
 - -            - linked list of material descriptions            (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Character Variable(s)
 - -   buffer - buffer for processing lines of output
 - -  ptr_buf - pointer to buffer
*/
  ascii_string buffer;
  char *ptr_buf;
/* - Structured Variable(s)
 - -  ptr - pointer to surface usage linked list
*/
  usage_list *ptr = ptr_material_usage;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Edit List of Universes Assigned to Unique Lattices - - - - - - -  */
  header();
  lines(4);
  fprintf(nout,"0Description of Materials used in the Problem\n");
  fprintf(nout,"0Index Label\n\n");
  while(ptr->next != NULL){
    lines(1);
```

```
    fprintf(nout,"  %3i  %s\n",ptr->index,ptr->label);
    ptr = ptr->next;
  }
/* - Last Member of Linked List */
  lines(1);
  fprintf(nout,"  %3i  %s\n",ptr->index,ptr->label);
}
```

Function edit_surfaces

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct su_list{
                       struct su_list *last;
                       int index;
                       ascii_string label;
                       ascii_string value;
                       char mnemonic[4];
                       ascii_string equivalent_label;
                       struct su_list *next;
                      } surface_usage_list;

void header();
void lines(int);
void bufferpad(char[],int,int);

void edit_surfaces(surface_usage_list *ptr_surface_usage){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * e d i t _ s u r f a c e s  *  edits the descriptions of surfaces
 - -                                 used in the problem
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -  surface_usage_list
 - -            - linked list of surface descriptions            (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - -  seq_num - sequence number for surfaces in surface definition
 - -            linked list
*/
  int seq_num = 0;
/* - Character Variable(s)
 - -   buffer - buffer for processing lines of output
 - -  ptr_buf - pointer to buffer
*/
  ascii_string buffer;
  char *ptr_buf;
/* - Structured Variable(s)
 - -  ptr - pointer to surface usage linked list
*/
  surface_usage_list *ptr = ptr_surface_usage;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Edit List of Universes Assigned to Unique Lattices - - - - - - -  */
  header();
  lines(7);
  fprintf(nout,"0Description of Surfaces used in the Problem\n");
  fprintf(nout,"0Seq. No. Index          Label\n");
  fprintf(nout,"                 Mnemonic Definition\n");
  fprintf(nout,"                          Equivalent Label\n\n");
  while(ptr->next != NULL){
    ptr_buf = buffer;
    seq_num++;
    ptr = ptr->next;
    if(strlen(ptr->equivalent_label) != 0)
      lines(3);
    else
      lines(2);
    fprintf(nout,"  %3i     %3i            %s\n",seq_num,ptr->index
     ,ptr->label);
    sprintf(buffer,"                   %s",ptr->mnemonic);
    ptr_buf = buffer;
    bufferpad(buffer,strlen(buffer),26);
    ptr_buf += 25*sizeof(char);
    sprintf(ptr_buf,"%s",ptr->value);
    fprintf(nout,"%s\n",buffer);
    if(strlen(ptr->equivalent_label) != 0)
      fprintf(nout,"                          %s\n"
       ,ptr->equivalent_label);
  }
}
```

Function edit_universes

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

void header();
void lines();

int search_fau_list(int,int,int *,int);

void edit_universes(int ncolcc,int nrowcc,int *ccmap,int *fau_fill
 ,int core_f,int ncolp,int nrowp,int nu_cc,int nlatticm,int nbundlm
 ,int *ptr_ufl,int *ptr_ufa){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * e d i t _ u n i v e r s e s  *  edits the universe indices for
 - -                                   control cells and fuel assemblies
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     ncolcc - Number of Columns in Control Cell Map          (input)
 - -     nrowcc - Number of Rows in Control Cell Map             (input)
 - -      ccmap - Map of Control Cell Universe Indices           (input)
```

```
 - -   fau_fill - Array Containing Fuel Assembly Assigments to   (input)
 - -              Unique Control Cells
 - -     core_f - Fraction of Core Modeled                       (input)
 - -              (1 - full core,
 - -               2 - half core,
 - -               4 - quarter core)
 - -      ncolp - number of columns in core map                  (input)
 - -      nrowp - number of rows in core map                     (input)
 - -      nu_cc - starting index for control cell indices        (input)
 - -   nlatticm - number of lattices with unique material defin- (input)
 - -              itions
 - -    nbundlm - number of fuel assemblies with unique material (input)
 - -              definitions
 - -    ptr_ufl - vector with universe indices for unique lat-   (input)
 - -              tice types
 - -    ptr_ufa - vector with universe indices for unique fuel   (input)
 - -              assembly types
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Varialble(s)
 - -        i - loop variable for columns in map
 - -        j - loop variable for rows in map
 - -        k - utility loop variable
 - -    n_row - index to row in fuel assembly map
 - -    n_col - index to column in fuel assembly map
 - -   holder - temporary storage for integer variable
 - -  ptr_map - pointer to ccmap array
 - -  ptr_fau - pointer to fau_fill array
 - -    value - value for fuel assembly universe index are returned
 - -            by search routine
*/
  short int i,j,k,n_row,n_col;
  int holder, value;
  int *ptr_map = ccmap, *ptr_fau = fau_fill;
/* - Character Variable(s)
 - -   buffer - buffer for processing lines of output
 - -  ptr_buf - pointer to buffer
*/
  ascii_string buffer;
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Edit List of Universes Assigned to Unique Lattices - - - - - - -  */
  header();
  lines(2);
  fprintf(nout," Universe Indices Corresponding to Unique Lattices\n\n");
  { short int min = 1,max = 30, ncycle;
    int *p = ptr_ufl;
    if(nlatticm < 30) max = nlatticm;
    ncycle = (nlatticm/30)+1;
    for(k = 1;k <= ncycle;k++){
```

```
      lines(2);
      for(i = 1;i <= max;i++) fprintf(nout," %3i",i);
      fprintf(nout,"\n");
      for(i = 1;i <= max;i++,p++) fprintf(nout," %3i",*p);
      fprintf(nout,"\n");
      max += 30;
      if(max > nlatticm) max = nlatticm;
    }
  }
  lines(1);
  fprintf(nout,"\n");
/* - Edit List of Universes Assigned to Unique Fuel Assemblies - - - - */
  lines(2);
  fprintf(nout," Universe Indices Corresponding to Unique Fuel");
  fprintf(nout," Assemblies\n\n");
  { short int min = 1, max = 30, ncycle;
    int *p = ptr_ufa;
    if(nbundlm < 30) max = nbundlm;
    ncycle = (nbundlm/30)+1;
    for(k = 1;k <= ncycle;k++){
      lines(2);
      for(i = 1;i <= max;i++) fprintf(nout," %3i",i);
      fprintf(nout,"\n");
      for(i = 1;i <= max;i++,p++) fprintf(nout," %3i",*p);
      fprintf(nout,"\n");
      max += 30;
      if(max > nbundlm) max = nbundlm;
    }
  }
  lines(1);
  fprintf(nout,"\n");
/* - Edit Map of Control Cell Universe Indices - - - - - - - - - - - - - */
  lines(4+nrowcc);
  fprintf(nout," Indicies for Control Cells\n\n");
  fprintf(nout," r/c");
  for(i = 1;i <= ncolcc;i++) fprintf(nout," %3i",i);
  fprintf(nout,"\n\n");
  for(j = 1;j <= nrowcc;j++){
    fprintf(nout," %3i",j);
    for(i = 1;i <= ncolcc;i++,ptr_map++)
      if(*ptr_map != 0)
        fprintf(nout," %3i",*ptr_map);
      else
        fprintf(nout,"    ");
    fprintf(nout,"\n");
  }
/* - Edit Map of Fuel Assembly Universe Indices - - - - - - - - - - - -  */
  lines(nrowp+5);
  fprintf(nout,"0Indices for Fuel Assemblies\n\n");
  fprintf(nout," r/c");
  for(i = 1;i <= ncolp;i++) fprintf(nout," %3i",i);
  fprintf(nout,"\n\n");
  ptr_map = ccmap;
/* - Process Quarter-core Geometry */
```

```
  if(core_f == 4){
/* - Process Top Row if Not Complete Control Cell */
    n_row = 1;
    if(nrowp%2 == 0){
      fprintf(nout," %3",n_row);
      for(i = 1;i < ncolcc;i++,ptr_map++){
        if((ncolp%2 == 0) && (i == 1)){
          if(*ptr_map != 0){
            value = search_fau_list(*ptr_map,4,fau_fill,nu_cc);
            if(value != 0)
              fprintf(nout," %3i",value);
            else
              fprintf(nout,"    ");
            value = search_fau_list(*ptr_map,3,fau_fill,nu_cc);
            if(value != 0)
              fprintf(nout," %3i",value);
            else
              fprintf(nout,"    ");
          }
          else
            fprintf(nout,"        ");
        }
        else{
          if(*ptr_map != 0){
            value = search_fau_list(*ptr_map,4,fau_fill,nu_cc);
            fprintf(nout," %3i",value);
          }
          else fprintf(nout,"    ");
        }
      }
      ptr_map++;
      value = search_fau_list(*ptr_map,4,fau_fill,nu_cc);
      if(value != 0)
        fprintf(nout," %3i",value);
      else
        fprintf(nout,"   ",value);
      fprintf(nout,"\n");
    }
/* - Process Bulk of Map */
    for(j = n_row;j < nrowcc;j++){
      fprintf(nout," %3i",((2*j)-1));
      sprintf(buffer," %3i",(2*j));
      ptr_buf = buffer;
      ptr_buf += 4*sizeof(char);
      for(i = 1;i < ncolcc;i++,ptr_map++){
        if((ncolp%2 == 0) && (i == 1)){
          if(*ptr_map != 0){
            value = search_fau_list(*ptr_map,2,fau_fill,nu_cc);
            if(value != 0)
              fprintf(nout," %3i",value);
            else
              fprintf(nout,"    ");
            value = search_fau_list(*ptr_map,3,fau_fill,nu_cc);
            if(value != 0)
```

```
            sprintf(ptr_buf," %3i",value);
          else
            strcat(buffer,"    ");
          ptr_buf += 4*sizeof(char);
        }
        else{
          fprintf(nout,"    ");
          strcat(buffer,"    ");
          ptr_buf += 4*sizeof(char);
        }
      }
      else{
        if(*ptr_map != 0){
          value = search_fau_list(*ptr_map,1,fau_fill,nu_cc);
          if(value != 0)
            fprintf(nout," %3i",value);
          else
            fprintf(nout,"    ");
          value = search_fau_list(*ptr_map,2,fau_fill,nu_cc);
          if(value != 0)
            fprintf(nout," %3i",value);
          else
            fprintf(nout,"    ");
          value = search_fau_list(*ptr_map,4,fau_fill,nu_cc);
          if(value != 0)
            sprintf(ptr_buf," %3i",value);
          else
            strcat(buffer,"    ");
          ptr_buf += 4*sizeof(char);
          value = search_fau_list(*ptr_map,3,fau_fill,nu_cc);
          if(value != 0)
            sprintf(ptr_buf," %3i",value);
          else
            strcat(buffer,"    ");
          ptr_buf += 4*sizeof(char);
        }
        else{
          fprintf(nout,"        ");
          strcat(buffer,"        ");
          ptr_buf += 8*sizeof(char);
        }
      }
    }
/* - Final Column */
    value = search_fau_list(*ptr_map,1,fau_fill,nu_cc);
    if(value != 0)
      fprintf(nout," %3i",value);
    else
      fprintf(nout,"    ");
    value = search_fau_list(*ptr_map,4,fau_fill,nu_cc);
    if(value != 0)
      sprintf(ptr_buf," %3i",value);
    else
      strcat(buffer,"    ");
```

```
      ptr_buf += 4*sizeof(char);
      fprintf(nout,"\n");
      fprintf(nout,buffer);
      fprintf(nout,"\n");
      ptr_map++;
    }
/* - Process Assemblies along Centerline */
    fprintf(nout," %3i",((2*nrowcc)-1));
    if(ncolp%2 != 0){
      sprintf(buffer," %3i",2*nrowcc);
      ptr_buf = buffer;
      ptr_buf += 4*sizeof(char);}
    for(i = 1;i < ncolcc;i++,ptr_map++){
      if(nrowp%2 != 0){
        value = search_fau_list(*ptr_map,1,fau_fill,nu_cc);
        if(value != 0)
          fprintf(nout," %3i",value);
        else
          fprintf(nout,"    ");
        value = search_fau_list(*ptr_map,2,fau_fill,nu_cc);
        if(value != 0)
          fprintf(nout," %3i",value);
        else
          fprintf(nout,"    ");}
      else{
        value = search_fau_list(*ptr_map,1,fau_fill,nu_cc);
        if(value != 0)
          fprintf(nout," %3i",value);
        else
          fprintf(nout,"    ");
        value = search_fau_list(*ptr_map,2,fau_fill,nu_cc);
        if(value != 0)
          fprintf(nout," %3i",value);
        else
          fprintf(nout,"    ");
        value = search_fau_list(*ptr_map,4,fau_fill,nu_cc);
        if(value != 0)
          sprintf(ptr_buf," %3i",value);
        else
          strcat(buffer,"    ");
        ptr_buf += 4*sizeof(char);
        value = search_fau_list(*ptr_map,3,fau_fill,nu_cc);
        if(value != 0)
          sprintf(ptr_buf," %3i",value);
        else
          strcat(buffer,"    ");
        ptr_buf += 4*sizeof(char);
      }
    }
    if(ncolp%2 != 0){
      value = search_fau_list(*ptr_map,1,fau_fill,nu_cc);
      if(value != 0)
        fprintf(nout," %3i",value);
      else
```

```
        fprintf(nout,"    ");
      fprintf(nout,"\n");}
    else{
      value = search_fau_list(*ptr_map,4,fau_fill,nu_cc);
      if(value != 0)
        fprintf(nout," %3i\n",value);
      else
        sprintf(buffer,"    ");
      fprintf(nout,"%s\n",buffer);
    }
  }
/* - Process Half-core Geometry */
  else if(core_f == 2){
    ;}
/* - Process Full-core Geometry */
  else{
    ;}
}
```

Function ge7x7_lattice

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                       } surface_usage_list;
```

```
void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
ll_material *material_match(a_record *,char[],float *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void abort();
void lines(int);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_like_but(char[],char[],FILE *,int *,char[],int,int *);

void ge7x7_lattice(float cthick,float asin,float wgap,float ngap
 ,float cradius,float fsrd,float cfsrd,float rpitch,float cod
 ,float cld,float pod,char frcmat[],char fcmat[],int latdim,int nft
 ,int *lattice,int *ncell,int *nmaterial,FILE *lu8,FILE *lu9
 ,FILE *lu10,surface_usage_list *ptr_surface_usage
 ,usage_list *ptr_material_usage,float inchannel_density
 ,float bypass_density,int *nuniverse,float *fp_density
 ,ascii_string matdsnam,ll_material *fmids,a_record *ptr_core_mtls
 ,int nlat,int *ufl,int *nsurface,char inchannel_material []){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e 7 x 7 _ l a t t i c e  *  Create Lattice Model for GE 7x7
 - -                                  Lattice Model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     cthick - Channel Thickness (cm)                          (input)
 - -       asin - Inner Span of Channel (cm)                      (input)
 - -       wgap - Wide Gap Thickness (cm)                         (input)
 - -       ngap - Narrow Gap Thickness (cm)                       (input)
 - -    cradius - Inner Radius of Channel Corner (cm)             (input)
 - -       fsrd - Clad Surface to Clad Surface Separation         (input)
 - -              (cm)
 - -      cfsrd - Clad Surface to Inner Channel Surface           (input)
 - -              Separation (cm)
 - -     rpitch - Fuel Rod Pitch (cm)                             (input)
 - -        cod - Cladding Outer Diameter (cm)                    (input)
 - -        cld - Cladding Thickness (cm)                         (input)
 - -        pod - Fuel Pellet Outer Diameter (cm)                 (input)
 - -     frcmat - Material Identifier for Fuel Rod Cladding       (input)
 - -      fcmat - Material Identifier for Channel                 (input)
 - -     latdim - Lattice Dimensionality                          (input)
 - -        nft - Number of Unique Fuel Rod Types                 (input)
 - -    lattice - Fuel Rod Type Map                               (input)
 - -      ncell - number of MCNP cells                              (i&o)
 - -  nmaterial - number of MCNP materials                          (i&o)
 - -        lu8 - stream pointer to scratch file for cell defin- (input)
 - -              itions
 - -        lu9 - stream pointer to scratch file for surface      (input)
 - -              definitions
 - -       lu10 - stream pointer to scratch file for material     (input)
```

```
 - -              definitions
 - - ptr_surface_usage
 - -            - list of defined surfaces                        (input)
 - - ptr_material_usage
 - -            - list of defined materials                       (input)
 - - inchannel_density
 - -            - density for in-channel moderator (g/cc)         (input)
 - - bypass_density
 - -            - density for bypass moderator (g/cc)             (input)
 - -  nuniverse - number of MCNP "universes"                        (i&o)
 - - fp_density - density for fuel pellets                        (input)
 - -   matdsnam - name for fuel intermediate material dataset     (input)
 - -      fmids - linked list for fuel material composition       (input)
 - - ptr_core_mtls
 - -            - pointer to linked list containing core mat-    (input)
 - -              erial definitions
 - -       nlat - index for unique lattice for which to create    (input)
 - -              MCNP lattice model
 - -        ufl - index for universe that contains the entire     (input)
 - -              lattice representation
 - -   nsurface - total number of surfaces                        (input)
 - - inchannel_material
 - -            - mnemonic for inchannel moderator material       (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         n - index for loops
 - -       len - length of ascii strings
 - -     index - utility variable for indices
 - - n_entries - number of entries in linked list for fuel material
 - -       ufr - universe index for first fuel rod
 - -      ufrl - universe index for fuel rod window lattice
 - -     uchan - universe index for channel and contents
 - -     cchan - cell index for channel
 - -      cwic - cell index for area inside channel
 - -  ncpellet - cell index for reference fuel pellet
 - -     ncgap - cell index for reference fuel/cladding gap
 - -    ncclad - cell index for reference cladding
 - -   ncwater - cell index for water outside of reference fuel rod
*/
  short int n;
  int len = 132, length, index;
  int n_entries;
  int ufr, ufrl, uchan;
  int cchan, cwic;
  int ncpellet, ncgap, ncclad, ncwater;
/* - Float Variable(s)
 - -    spor - reference fuel pellet outer surface
 - -    scir - reference cladding inner surface
 - -    scor - reference cladding outer surface
 - - xminfrw - XMIN surface for fuel rod window
 - - xmaxfrw - XMAX surface for fuel rod window
 - - yminfrw - XMIN surface for fuel rod window
```

```
 - - ymaxfrw - YMAX surfacr for fuel rod window
 - -  wgcowx - wide gap, channel outside wall, px surface
 - -  wgciwx - wide gap, channel inside wall, px surface
 - -  ngciwx - narrow gap, channel inside wall, px surface
 - -  ngcowx - narrow gap, channel outside wall, px surface
 - -  wgcowy - wide gap, channel outside wall, py surface
 - -  wgciwy - wide gap, channel inside wall, py surface
 - -  ngciwy - narrow gap, channel inside wall, py surface
 - -  ngcowy - narrow gap, channel outside wall, py surface
 - - xambig1 - ambiguity surface for channel corners (wide gap)
 - - xambig2 - ambiguity surface for channel corners (narrow gap)
 - - yambig1 - ambiguity surface for channel corners (wide gap)
 - - yambig2 - ambiguity surface for channel corners (narrow gap)
 - -    ccro - outer radius for channel corner
 - -    ccri - innner radius for channel corner
 - -    cc1x - center for channel corner 1
 - -    cc1y - center for channel corner 1
 - -    cc2x - center for channel corner 2
 - -    cc2y - center for channel corner 2
 - -    cc3x - center for channel corner 3
 - -    cc3y - center for channel corner 3
 - -    cc4x - center for channel corner 4
 - -    cc4y - center for channel corner 4
 - - density - density for material
 - - x_ofset - x-offset for fuel rod centering to base lattice position
 - - y_ofset - y-offset for fuel rod centering to base lattice position
*/
  float spor, scir, scor,xminfrw,xmaxfrw,yminfrw,ymaxfrw,wgcowx,wgciwx
   ,ngciwx,ngcowx,wgcowy,wgciwy,ngciwy,ngcowy,xambig1,xambig2,yambig1
   ,yambig2,ccro,ccri,cc1x,cc1y,cc2x,cc2y,cc3x,cc3y,cc4x,cc4y;
  float density,x_ofset,y_ofset;
/* - Character Variable(s)
 - -    label - label for cell or surface
 - - material - label for cell material for output edit
 - - mnemonic - MCNP mnemonic for surface definition
 - -    value - surface defintion values in string form
 - -   mdsnam - working pointer for lattice material intermediate
 - -           dataset name
*/
  char mnemonic[4];
  ascii_string label,material,value;
/* - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition linked list
*/
  ll_material *ptr_mtl;
/* - Compute Surfaces for Lattice - - - - - - - - - - - - - - - - - - - - */
  spor = pod/2;
  scir = (cod/2)-cld;
  scor = cod/2;
  xminfrw = -rpitch/2;
```

```
  xmaxfrw = rpitch/2;
  yminfrw = -rpitch/2;
  ymaxfrw = rpitch/2;
  wgcowx = wgap;
  wgciwx = wgap+cthick;
  ngciwx = wgap+cthick+asin;
  ngcowx = wgap+(2*cthick)+asin;
  wgcowy = -wgap;
  wgciwy = -(wgap+cthick);
  ngciwy = -(wgap+cthick+asin);
  ngcowy = -(wgap+(2*cthick)+asin);
  xambig1 = wgap+cthick+cradius;
  xambig2 = wgap+cthick+asin-cradius;
  yambig1 = -(wgap+cthick+cradius);
  yambig2 = -(wgap+cthick+asin-cradius);
  ccro = cradius+cthick;
  ccri = cradius;
  cc1x = wgap+cthick+cradius;
  cc1y = -(wgap+cthick+cradius);
  cc2x = wgap+cthick+asin-cradius;
  cc2y = -(wgap+cthick+cradius);
  cc3x = wgap+cthick+asin-cradius;
  cc3y = -(wgap+cthick+asin-cradius);
  cc4x = wgap+cthick+cradius;
  cc4y = -(wgap+cthick+asin-cradius);
/* - compute offset to move fuel rod to reference location */
  x_ofset = wgap+cthick+cfsrd+(cod/2);
  y_ofset = -x_ofset;
/* - Build Lattice Model Starting with Fuel Pellet - - - - - - - - - - */
  for( n = 0; n < nft; n++){
    (*nuniverse)++;(*nmaterial)++;
    if(n == 0) ufr = *nuniverse;
/* - Pellet Cell */
    sprintf(label,"Fuel Pellet, #%i, L%i",(n+1),nlat);
    sprintf(material,"%s (%i)",matdsnam,(n+1));
    search_usage_list(1,material,&index,ptr_material_usage);
    if(n == 0){
      if(index == 0){
        add_cell(label,material,lu8,ncell,*nmaterial,-(*fp_density)
         ,nuniverse);
        n_entries = 1;}
      else{
        add_cell(label,material,lu8,ncell,index,-(*fp_density)
         ,nuniverse);
        n_entries = 0;}
      ncpellet = *ncell;}
    else{
      if(index == 0){
        sprintf(value,"mat= %i rho= %10.4E"
         ,*nmaterial,-(*fp_density));
        n_entries = 1;}
      else{
        sprintf(value,"mat= %i rho= %10.4E"
         ,index,-(*fp_density));
```

```
        n_entries = 0;}
      add_like_but(label,material,lu8,ncell,value,ncpellet
       ,nuniverse);}
    { usage_list *ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(material,index,ptr_ml);
    }
    fp_density++;
/* - Pellet Surface */
    if(n == 0){
      strcpy(label,"");
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i"
       ,(n+1),nlat);
      strcpy(mnemonic,"c/z");
      strcpy(value,"");
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,spor);
      index = -1;
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Pellet Materials */
    sprintf(label,"%s (%i)",matdsnam,(n+1));
/* - Determine number of entries in Linked List */
    if(n_entries == 1){
      { ll_material *f = fmids;
        do{
          f = f->next;
          n_entries++;}
        while(f->next != NULL);
      }
      add_material(lu10,nmaterial,label,n_entries,fmids);
    }
    if(n < (nft+1)) fmids++;
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Fuel/Cladding Gap - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Fuel/Cladding Gap, #%i, L%i",(n+1),nlat);
    strcpy(material,"void");
    if(n == 0){
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
      ncgap = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncgap,nuniverse);
/* - Surfaces */
    if(n == 0){
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
```

```
        ," Surface Not Found in Linked List, label = %s\n",label);
       abort();}
     add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
      ,nsurface);
     sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
     strcpy(mnemonic,"c/z");
     index = -1;
     sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scir);
     add_surface(0,label,mnemonic,lu8,lu9,&index,value
      ,ptr_surface_usage,nsurface);
   }
/* - Material -- void used */
   fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Cladding - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
   sprintf(label,"Cladding, #%i, L%i",(n+1),nlat);
   strcpy(material,frcmat);
   if(n == 0){
     search_usage_list(1,frcmat,&index,ptr_material_usage);
     n_entries = 0;
     if(index == 0){
       { usage_list *ptr_ml;
         ptr_mtl =
          material_match(ptr_core_mtls,frcmat,&density,&n_entries);
         (*nmaterial)++;
         ptr_ml = ptr_material_usage;
         while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
         index = (ptr_ml->index)+1;
         ptr_ml = load_usage_list(frcmat,index,ptr_ml);
       }
     }
     else{
       (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
       n_entries = 0;}
     add_cell(label,material,lu8,ncell,index,-density,nuniverse);
     ncclad = *ncell;}
   else
     add_like_but(label,material,lu8,ncell,"",ncclad,nuniverse);
/* - Surface(s) */
   if(n == 0){
     sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
     search_surface_usage_list(1,label,&index,"","",""
      ,ptr_surface_usage);
     if(index == 0){
       lines(3);
       fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
       fprintf(nout," ge7x7_lattice --\n");
       fprintf(nout
        ," Surface Not Found in Linked List, label = %s\n",label);
       abort();}
     add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
      ,nsurface);
     sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
     strcpy(mnemonic,"c/z");
```

```
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scor);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
     }
/* - Material */
    if(n == 0){
      if(n_entries != 0){
        add_material(lu10,nmaterial,frcmat,n_entries,ptr_mtl);
        rollup_llm(ptr_mtl);
        n_entries = 0;}
    }
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Water Outside of Outer Surface of Cladding - - - - - - - - - - -  */
/* - Cell */
    sprintf(label,"Water Outside Cladding #%i, L%i",(n+1),nlat);
    strcpy(material,inchannel_material);
    if(n == 0){
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Material Not Found in Linked List, material = %s\n",label);
        abort();}
      add_cell(label,material,lu8,ncell,index,-inchannel_density
       ,nuniverse);
      ncwater = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncwater,nuniverse);
/* - Surface */
    if(n == 0){
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","","
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
/* - Materials */
/* - In-channel Material should already exist as a material */
    if(n == 0) fprintf(lu8,"\n");
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
  }
/* - Create Lattice for Fuel Assembly and Populate with Previously
 - - Defined Fuel Rod Universes - - - - - - - - - - - - - - - - - - -  */
/* - Cell */
```

```
  sprintf(label,"Reference Fuel Rod Cell, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  (*nuniverse)++;
  ufrl = *nuniverse;
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"XMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",(xmaxfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"XMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",(xminfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",(yminfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",(ymaxfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8," lat= 1 u = %i imp:n= 1.0\n",*nuniverse);
  fprintf(lu8,"      fill= -1:%i -1:%i 0:0\n"
   ,latdim,latdim);
  lines(latdim+1);
  fprintf(nout,"     Universes Filling the Lattice:\n");
  { short int i,j;
    int *plattice = lattice;
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
    for(i = 1;i <= latdim;i++){
      fprintf(lu8,"        %3i ",*nuniverse);
      fprintf(nout,"      ");
      for(j = 1;j <= latdim;j++,plattice++){
        fprintf(lu8,"%3i ",(*plattice+ufr-1));
```

```
        fprintf(nout,"%3i ",(*plattice+ufr-1));}
      fprintf(lu8,"%3i\n",*nuniverse);
      fprintf(nout,"\n");}
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
  }
/* - Create Channel Model and Populate with Lattice - - - - - - - -  */
/* - Channel - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*nuniverse)++;
  uchan = *nuniverse;
  sprintf(label,"Channel, L%i",nlat);
  search_usage_list(1,fcmat,&index,ptr_material_usage);
  n_entries = 0;
  if(index == 0){
    { usage_list *ptr_ml;
      ptr_mtl =
       material_match(ptr_core_mtls,fcmat,&density,&n_entries);
      (*nmaterial)++;
      ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(fcmat,index,ptr_ml);
    }
  }
  else{
    (void) material_match(ptr_core_mtls,fcmat,&density,&n_entries);
    n_entries = 0;
  }
  add_cell(label,fcmat,lu8,ncell,index,-density,nuniverse);
  cchan = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",wgcowx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",wgciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",yambig1);
```

```
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",yambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,"):(");
sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",wgcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",wgciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = 1;
sprintf(value,"%10.4E",xambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",xambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",ngciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
,nlat);
strcpy(mnemonic,"py");
index = 1;
```

```
sprintf(value,"%10.4E",ngcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
strcpy(mnemonic,"px");
sprintf(value,"%10.4E",ngciwx);
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
 ,nlat);
strcpy(mnemonic,"px");
sprintf(value,"%10.4E",ngcowx);
index = -1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","
 ,ptr_surface_usage);
if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Channel Corners */
/* - Northwest Corner */
  sprintf(label,"Channel Corner Outer Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cclx,ccly,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cclx,ccly,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Northeast Corner */
  sprintf(label,"Channel Corner Outer Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Southeast Corner */
  sprintf(label,"Channel Corner Outer Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Southwest Corner */
```

```
  sprintf(label,"Channel Corner Outer Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",uchan);
/* - Materials */
  if(n_entries != 0){
    add_material(lu10,nmaterial,fcmat,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
    n_entries = 0;}
/* - Area Inside the Channel - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Active Fuel Area, L%i",nlat);
```

```
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density,nuniverse);
  cwic = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
```

```
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n      (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
```

```
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Channel Corner Inner Radius (NW), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
```

```
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
  sprintf(label
   ,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
```

```
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8,"):(");
 sprintf(label
  ,"Channel Corner Inner Radius (SE), L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
```

```
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
  sprintf(label
   ,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      fill= %i u= %i imp:n= 1.0\n",ufrl,uchan);
  lines(1);
  fprintf(nout,"      Filled with Universe %i\n",ufrl);
/* - Area Outside the Channel - - - - - - - - - - - - - - - - - - - - -  */
/* - Cell */
  sprintf(label,"Water Outside of Channel, L%i",nlat);
```

```
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Regions Outside of Four Channel Walls */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
```

```
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
```

```
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
/* - Assigned Universe Number to Return Variable */
  *ufl = uchan;
}
```

Function ge8x8_lattice_swr

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;
```

```
typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                       } surface_usage_list;

typedef struct fuel_geometry{
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              char frcmat[6];
                              char fcmat[6];
                             } fg_list;


void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
ll_material *material_match(a_record *,char[],float *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void abort();
void lines(int);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_like_but(char[],char[],FILE *,int *,char[],int,int *);

void ge8x8_lattice_swr(float cthick,float asin,float wgap,float ngap
 ,float cradius,float fsrd,float cfsrd,float rpitch,float cod
 ,float cld,float pod,char frcmat[],char fcmat[],int latdim,int nft
```

```
,int *lattice,int *ncell,int *nmaterial,FILE *lu8,FILE *lu9
,FILE *lu10,surface_usage_list *ptr_surface_usage
,usage_list *ptr_material_usage,float inchannel_density
,float bypass_density,int *nuniverse,float *fp_density
,ascii_string matdsnam,ll_material *fmids,a_record *ptr_core_mtls
,int nlat,int *ufl,int *nsurface,char inchannel_material []){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e 8 x 8 _ l a t t i c e _ s w r  *  Create Lattice Model for
 - -                                          GE 8x8 Lattice with Water
 - -                                          Rods the same Size as the
 - -                                          Fuel Rods
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     cthick - Channel Thickness (cm)                          (input)
 - -       asin - Inner Span of Channel (cm)                      (input)
 - -       wgap - Wide Gap Thickness (cm)                         (input)
 - -       ngap - Narrow Gap Thickness (cm)                       (input)
 - -    cradius - Inner Radius of Channel Corner (cm)             (input)
 - -       fsrd - Clad Surface to Clad Surface Separation         (input)
 - -              (cm)
 - -      cfsrd - Clad Surface to Inner Channel Surface           (input)
 - -              Separation (cm)
 - -     rpitch - Fuel Rod Pitch (cm)                             (input)
 - -        cod - Cladding Outer Diameter (cm)                    (input)
 - -        cld - Cladding Thickness (cm)                         (input)
 - -        pod - Fuel Pellet Outer Diameter (cm)                 (input)
 - -     frcmat - Material Identifier for Fuel Rod Cladding       (input)
 - -      fcmat - Material Identifier for Channel                 (input)
 - -     latdim - Lattice Dimensionality                          (input)
 - -        nft - Number of Unique Fuel Rod Types                 (input)
 - -    lattice - Fuel Rod Type Map                               (input)
 - -      ncell - number of MCNP cells                              (i&o)
 - -  nmaterial - number of MCNP materials                          (i&o)
 - -        lu8 - stream pointer to scratch file for cell defin- (input)
 - -              itions
 - -        lu9 - stream pointer to scratch file for surface      (input)
 - -              definitions
 - -       lu10 - stream pointer to scratch file for material     (input)
 - -              definitions
 - - ptr_surface_usage
 - -            - list of defined surfaces                       (input)
 - - ptr_material_usage
 - -            - list of defined materials                      (input)
 - - inchannel_density
 - -            - density for in-channel moderator (g/cc)        (input)
 - - bypass_density
 - -            - density for bypass moderator (g/cc)            (input)
 - -  nuniverse - number of MCNP "universes"                        (i&o)
 - - fp_density - density for fuel pellets                        (input)
 - -   matdsnam - name for fuel intermediate material dataset     (input)
 - -      fmids - linked list for fuel material composition       (input)
 - - ptr_core_mtls
 - -            - pointer to linked list containing core mat-    (input)
 - -              erial definitions
```

```
 - -       nlat - index for unique lattice for which to create   (input)
 - -              MCNP lattice model
 - -        ufl - index for universe that contains the entire    (input)
 - -              lattice representation
 - -   nsurface - total number of surfaces                       (input)
 - - inchannel_material
 - -            - mnemonic for inchannel moderator material      (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         n - index for loops
 - -       len - length of ascii strings
 - -     index - utility variable for indices
 - - n_entries - number of entries in linked list for fuel material
 - -       ufr - universe index for first fuel rod
 - -      ufrl - universe index for fuel rod window lattice
 - -     uchan - universe index for channel and contents
 - -     cchan - cell index for channel
 - -      cwic - cell index for area inside channel
 - -  ncpellet - cell index for reference fuel pellet
 - -     ncgap - cell index for reference fuel/cladding gap
 - -    ncclad - cell index for reference cladding
 - -   ncwater - cell index for water outside of reference fuel rod
 - -       uwr - universe index for water rod
*/
  short int n;
  int len = 132, length, index;
  int n_entries;
  int ufr, ufrl, uchan, uwr;
  int cchan, cwic;
  int ncpellet, ncgap, ncclad, ncwater;
/* - Float Variable(s)
 - -    spor - reference fuel pellet outer surface
 - -    scir - reference cladding inner surface
 - -    scor - reference cladding outer surface
 - - xminfrw - XMIN surface for fuel rod window
 - - xmaxfrw - XMAX surface for fuel rod window
 - - yminfrw - XMIN surface for fuel rod window
 - - ymaxfrw - YMAX surfacr for fuel rod window
 - -  wgcowx - wide gap, channel outside wall, px surface
 - -  wgciwx - wide gap, channel inside wall, px surface
 - -  ngciwx - narrow gap, channel inside wall, px surface
 - -  ngcowx - narrow gap, channel outside wall, px surface
 - -  wgcowy - wide gap, channel outside wall, py surface
 - -  wgciwy - wide gap, channel inside wall, py surface
 - -  ngciwy - narrow gap, channel inside wall, py surface
 - -  ngcowy - narrow gap, channel outside wall, py surface
 - - xambig1 - ambiguity surface for channel corners (wide gap)
 - - xambig2 - ambiguity surface for channel corners (narrow gap)
 - - yambig1 - ambiguity surface for channel corners (wide gap)
 - - yambig2 - ambiguity surface for channel corners (narrow gap)
 - -    ccro - outer radius for channel corner
 - -    ccri - innner radius for channel corner
```

```
 - -    cc1x - center for channel corner 1
 - -    cc1y - center for channel corner 1
 - -    cc2x - center for channel corner 2
 - -    cc2y - center for channel corner 2
 - -    cc3x - center for channel corner 3
 - -    cc3y - center for channel corner 3
 - -    cc4x - center for channel corner 4
 - -    cc4y - center for channel corner 4
 - - density - density for material
 - - x_ofset - x-offset for fuel rod centering to base lattice position
 - - y_ofset - y-offset for fuel rod centering to base lattice position
*/
  float spor, scir, scor,xminfrw,xmaxfrw,yminfrw,ymaxfrw,wgcowx,wgciwx
   ,ngciwx,ngcowx,wgcowy,wgciwy,ngciwy,ngcowy,xambig1,xambig2,yambig1
   ,yambig2,ccro,ccri,cc1x,cc1y,cc2x,cc2y,cc3x,cc3y,cc4x,cc4y;
  float density,x_ofset,y_ofset;
/* - Character Variable(s)
 - -    label - label for cell or surface
 - - material - label for cell material for output edit
 - - mnemonic - MCNP mnemonic for surface definition
 - -    value - surface defintion values in string form
 - -   mdsnam - working pointer for lattice material intermediate
 - -           dataset name
*/
  char mnemonic[4];
  ascii_string label,material,value;
/* - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition linked list
*/
  ll_material *ptr_mtl;
/* - Compute Surfaces for Lattice - - - - - - - - - - - - - - - - - - - - */
  spor = pod/2;
  scir = (cod/2)-cld;
  scor = cod/2;
  xminfrw = -rpitch/2;
  xmaxfrw = rpitch/2;
  yminfrw = -rpitch/2;
  ymaxfrw = rpitch/2;
  wgcowx = wgap;
  wgciwx = wgap+cthick;
  ngciwx = wgap+cthick+asin;
  ngcowx = wgap+(2*cthick)+asin;
  wgcowy = -wgap;
  wgciwy = -(wgap+cthick);
  ngciwy = -(wgap+cthick+asin);
  ngcowy = -(wgap+(2*cthick)+asin);
  xambig1 = wgap+cthick+cradius;
  xambig2 = wgap+cthick+asin-cradius;
  yambig1 = -(wgap+cthick+cradius);
  yambig2 = -(wgap+cthick+asin-cradius);
```

```
  ccro = cradius+cthick;
  ccri = cradius;
  cc1x = wgap+cthick+cradius;
  cc1y = -(wgap+cthick+cradius);
  cc2x = wgap+cthick+asin-cradius;
  cc2y = -(wgap+cthick+cradius);
  cc3x = wgap+cthick+asin-cradius;
  cc3y = -(wgap+cthick+asin-cradius);
  cc4x = wgap+cthick+cradius;
  cc4y = -(wgap+cthick+asin-cradius);
/* - compute offset to move fuel rod to reference location */
  x_ofset = wgap+cthick+cfsrd+(cod/2);
  y_ofset = -x_ofset;
/* - Build Lattice Model Starting with Fuel Pellet - - - - - - - - - - */
  for( n = 0; n < nft; n++){
    (*nuniverse)++;(*nmaterial)++;
    if(n == 0) ufr = *nuniverse;
/* - Pellet Cell */
    sprintf(label,"Fuel Pellet, #%i, L%i",(n+1),nlat);
    sprintf(material,"%s (%i)",matdsnam,(n+1));
    search_usage_list(1,material,&index,ptr_material_usage);
    if(n == 0){
      if(index == 0){
        add_cell(label,material,lu8,ncell,*nmaterial,-(*fp_density)
         ,nuniverse);
        n_entries = 1;}
      else{
        add_cell(label,material,lu8,ncell,index,-(*fp_density)
         ,nuniverse);
        n_entries = 0;}
      ncpellet = *ncell;}
    else{
      if(index == 0){
        sprintf(value,"mat= %i rho= %10.4E"
         ,*nmaterial,-(*fp_density));
        n_entries = 1;}
      else{
        sprintf(value,"mat= %i rho= %10.4E"
         ,index,-(*fp_density));
        n_entries = 1;}
      add_like_but(label,material,lu8,ncell,value,ncpellet
       ,nuniverse);}
    { usage_list *ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(material,index,ptr_ml);
    }
    fp_density++;
/* - Pellet Surface */
    if(n == 0){
      strcpy(label,"");
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i"
       ,(n+1),nlat);
      strcpy(mnemonic,"c/z");
```

```
      strcpy(value,"");
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,spor);
      index = -1;
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Pellet Materials */
    sprintf(label,"%s (%i)",matdsnam,(n+1));
/* - Determine number of entries in Linked List */
    if(n_entries == 1){
      { ll_material *f = fmids;
        do{
          f = f->next;
          n_entries++;}
        while(f->next != NULL);
      }
      add_material(lu10,nmaterial,label,n_entries,fmids);
    }
    if(n < (nft+1)) fmids++;
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Fuel/Cladding Gap - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Fuel/Cladding Gap, #%i, L%i",(n+1),nlat);
    strcpy(material,"void");
    if(n == 0){
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
      ncgap = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncgap,nuniverse);
/* - Surfaces */
    if(n == 0){
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scir);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material -- void used */
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Cladding - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
```

```
    sprintf(label,"Cladding, #%i, L%i",(n+1),nlat);
    strcpy(material,frcmat);
    if(n == 0){
      search_usage_list(1,frcmat,&index,ptr_material_usage);
      n_entries = 0;
      if(index == 0){
        { usage_list *ptr_ml;
          ptr_mtl =
           material_match(ptr_core_mtls,frcmat,&density,&n_entries);
          (*nmaterial)++;
          ptr_ml = ptr_material_usage;
          while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
          index = (ptr_ml->index)+1;
          ptr_ml = load_usage_list(frcmat,index,ptr_ml);
        }
      }
      else{
        (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
        n_entries = 0;}
      add_cell(label,material,lu8,ncell,index,-density,nuniverse);
      ncclad = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncclad,nuniverse);
/* - Surface(s) */
    if(n == 0){
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scor);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
     }
/* - Material */
    if(n == 0){
      if(n_entries != 0){
        add_material(lu10,nmaterial,frcmat,n_entries,ptr_mtl);
        rollup_llm(ptr_mtl);
        n_entries = 0;}
    }
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Water Outside of Outer Surface of Cladding - - - - - - - - - - - - */
/* - Cell */
```

```
    sprintf(label,"Water Outside Cladding #%i, L%i",(n+1),nlat);
    strcpy(material,inchannel_material);
    if(n == 0){
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Material Not Found in Linked List, material = %s\n"
         ,material);
        abort();}
      add_cell(label,material,lu8,ncell,index,-inchannel_density
       ,nuniverse);
      ncwater = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncwater,nuniverse);
/* - Surface */
    if(n == 0){
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
/* - Materials */
/* - In-channel Material should already exist as a material */
    if(n == 0) fprintf(lu8,"\n");
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
  }
/* - Create Water Rod for Lattice - - - - - - - - - - - - - - - - - -   */
/* - Water */
  (*nuniverse)++;
  uwr = *nuniverse;
  sprintf(label,"Water Rod Center, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
```

```
  sprintf(label,"Fuel Cladding Inner Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Cladding */
  sprintf(label,"Water Rod Cladding, L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Fuel Cladding Inner Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Fuel Cladding Outer Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
```

```
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Fuel Cladding Outer Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Create Lattice for Fuel Assembly and Populate with Previously
 - - Defined Fuel Rod Universes - - - - - - - - - - - - - - - - - - - -   */
/* - Cell */
  sprintf(label,"Reference Fuel Rod Cell, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  (*nuniverse)++;
  ufrl = *nuniverse;
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"XMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",(xmaxfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
```

```
  sprintf(label,"XMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",(xminfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",(yminfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",(ymaxfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8," lat= 1 u = %i imp:n= 1.0\n",*nuniverse);
  fprintf(lu8,"      fill= -1:%i -1:%i 0:0\n"
   ,latdim,latdim);
  lines(latdim+1);
  fprintf(nout,"     Universes Filling the Lattice:\n");
  { short int i,j;
    int *plattice = lattice;
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
    for(i = 1;i <= latdim;i++){
      fprintf(lu8,"        %3i ",*nuniverse);
      fprintf(nout,"      ");
      for(j = 1;j <= latdim;j++,plattice++){
        if(*plattice > 0){
          fprintf(lu8,"%3i ",(*plattice+ufr-1));
          fprintf(nout,"%3i ",(*plattice+ufr-1));}
        else{
          fprintf(lu8,"%3i ",uwr);
          fprintf(nout,"%3i ",uwr);}
      }
      fprintf(lu8,"%3i\n",*nuniverse);
      fprintf(nout,"\n");}
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
  }
/* - Create Channel Model and Populate with Lattice - - - - - - - -  */
/* - Channel - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*nuniverse)++;
  uchan = *nuniverse;
  sprintf(label,"Channel, L%i",nlat);
  search_usage_list(1,fcmat,&index,ptr_material_usage);
  n_entries = 0;
  if(index == 0){
```

```
    { usage_list *ptr_ml;
      ptr_mtl =
       material_match(ptr_core_mtls,fcmat,&density,&n_entries);
      (*nmaterial)++;
      ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(fcmat,index,ptr_ml);
    }
  }
  else{
    (void) material_match(ptr_core_mtls,fcmat,&density,&n_entries);
    n_entries = 0;
  }
  add_cell(label,fcmat,lu8,ncell,index,-density,nuniverse);
  cchan = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",wgcowx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",wgciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",yambig1);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",yambig2);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"):(");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",wgcowy);
```

```
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",wgciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = 1;
sprintf(value,"%10.4E",xambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",xambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",ngciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",ngcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
strcpy(mnemonic,"px");
sprintf(value,"%10.4E",ngciwx);
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
 ,nlat);
strcpy(mnemonic,"px");
sprintf(value,"%10.4E",ngcowx);
index = -1;
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Channel Corners */
/* - Northwest Corner */
  sprintf(label,"Channel Corner Outer Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cclx,ccly,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cclx,ccly,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
```

```
  fprintf(lu8,"):(");
/* - Northeast Corner */
  sprintf(label,"Channel Corner Outer Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Southeast Corner */
  sprintf(label,"Channel Corner Outer Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
```

```
  sprintf(label,"Channel Corner Inner Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Southwest Corner */
  sprintf(label,"Channel Corner Outer Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
```

```
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",uchan);
/* - Materials */
  if(n_entries != 0){
    add_material(lu10,nmaterial,fcmat,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
    n_entries = 0;}
/* - Area Inside the Channel - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Active Fuel Area, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density,nuniverse);
  cwic = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
```

```
search_surface_usage_list(1,label,&index,"","","",
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n      (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
```

```
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Channel Corner Inner Radius (NW), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
```

```
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label
 ,"Channel Corner Inner Radius (NE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Channel Corner Inner Radius (SE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",""
 ,ptr_surface_usage);
if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
  sprintf(label
   ,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      fill= %i u= %i imp:n= 1.0\n",ufrl,uchan);
  lines(1);
  fprintf(nout,"     Filled with Universe %i\n",ufrl);
/* - Area Outside the Channel - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Water Outside of Channel, L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Regions Outside of Four Channel Walls */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
```

```
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8," : ");
sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8," : ");
sprintf(label,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8," : ");
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
```

```
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
```

```
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
/* - Assigned Universe Number to Return Variable */
  *ufl = uchan;
}
```

Function generate_deck

```
#include<stdio.h>

void generate_deck(FILE *lmcnp,FILE *lu8,FILE *lu9,FILE *lu10){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * g e n e r a t e  d e c k  *  Generates MCNP Input Deck
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      lmcnp - pointer to MCNP input file                    (input)
 - -        lu8 - pointer to scratch file for cell representa-  (input)
 - -              tations
 - -        lu9 - pointer to scratch file for surface repre-    (input)
 - -              tations
 - -       lu10 - pointer to scratch file for material repre-   (input)
 - -              sentations
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
/* - Copy Contents of Scratch File for Cell Representations - - - - - */
  copy_ascii_file(lmcnp,lu8);
/* - Insert Blank Separator Line */
  fprintf(lmcnp,"\n");
/* - Copy Contents of Scratch File for Surface Representations - - - */
  copy_ascii_file(lmcnp,lu9);
/* - Insert Blank Separator Line */
  fprintf(lmcnp,"\n");
/* - Copy Contents of Scratch File for Material Representations - - - */
  copy_ascii_file(lmcnp,lu10);
  fprintf(lmcnp,"print\n");
/* - Insert Blank Separator Line */
  fprintf(lmcnp,"\n");
  return;
}
```

Function generate_lattice_model

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                            struct ascii_record *last;
                            ascii_string line;
                            struct ascii_record *next;
                           } a_record;
typedef struct s_material{
                          struct s_material *last;
                          int atomic_number;
                          int mass_number;
                          float weight_percentage;
                          char library_suffix[5];
                          struct s_material *next;
                         } ll_material;
typedef struct u_list {
                       struct u_list *last;
                       int index;
                       ascii_string label;
                       struct u_list *next;
                      } usage_list;

typedef struct su_list {
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;

typedef struct fuel_geometry {
                              ascii_string gds_name;
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              char frcmat[6];
                              char fcmat[6];
                             } fg_list;
```

```
void ge7x7_lattice(float,float,float,float,float,float,float,float
 ,float,float,float,char[],char[],int,int,int *,int *,int *,FILE *
 ,FILE *,FILE *,surface_usage_list *,usage_list *,float,float,int *
 ,float *,ascii_string,ll_material *,a_record *,int,int *,int *
 ,char[]);
void ge8x8_lattice_swr(float,float,float,float,float,float,float,float
 ,float,float,float,char[],char[],int,int,int *,int *,int *,FILE *
 ,FILE *,FILE *,surface_usage_list *,usage_list *,float,float,int *
 ,float *,ascii_string,ll_material *,a_record *,int,int *,int *
 ,char[]);
void rollup_llm(ll_material *);
int *memory_integer(int,int,int *);
ll_material *load_fuel_material(int *,char[],char[],int,int **
 ,float **);
int mchar(int *,char[]);

void generate_lattice_model(int *ncell,int *nmaterial,FILE *lu8
 ,FILE *lu9,FILE *lu10,surface_usage_list *ptr_surface_usage
 ,usage_list *ptr_material_usage,int *ptr_correspondence_table
 ,fg_list *ptr_lg_ds,ascii_string *lmdsnam,int nlat,char fprefix[]
 ,float inchannel_density,float bypass_density,int *nuniverse
 ,a_record *ptr_core_mtls,int *ufl,int *nsurface
 ,char inchannel_material[]){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e n e r a t e _ l a t t i c e _ m o d e l  *  Create Lattice
 - -                                                   Model for a
 - -                                                   Unique Node
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      ncell - number of MCNP cells                                (i&o)
 - -   nmaterial - number of MCNP materials                          (i&o)
 - -        lu8 - stream pointer to scratch file for cell defin-  (input)
 - -              itions
 - -        lu9 - stream pointer to scratch file for surface      (input)
 - -              definitions
 - -       lu10 - stream pointer to scratch file for material     (input)
 - -              definitions
 - - ptr_surface_usage
 - -            - list of defined surfaces                       (input)
 - - ptr_material_usage
 - -            - list of defined materials                      (input)
 - - ptr_correspondence_table
 - -            - table relating lattices with unique material   (input)
 - -              definitions with geometry definitions
 - -  ptr_lg_ds - list of lattice geometry datasets              (input)
 - -    lmdsnam - list of material intermediate dataset names    (input)
 - -       nlat - index for unique lattice for which to create   (input)
 - -              MCNP lattice model
 - -    fprefix - prefix for fuel material intermediate datasets (input)
 - - inchannel_density
 - -            - density for in-channel moderator (g/cc)        (input)
 - - bypass_density
 - -            - density for bypass moderator (g/cc)            (input)
```

```
 - -  nuniverse - number of MCNP "universes"                        (i&o)
 - - ptr_core_mtls
 - -            - pointer to linked list containing core mat-     (input)
 - -              erial definitions
 - -        ufl - universe definition for unique lattices          (output)
 - -   nsurface - total number of surfaces                          (i&o)
 - - inchannel_material
 - -            - menmonic for inchannel moderator material        (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  latdim - lattice dimensionality
 - -     nwr - number of water rods
 - -   nglat - index for lattice geometry dataset corresponding to
 - -           lattice material dataset
 - - lattice - fuel rod type map for lattice
 - -     nft - number of fuel rod types
*/
  int latdim, nwr;
  int nglat, *lattice, nft;
/* - Float Variable(s)
 - -     cthick - Channel Thickness (cm)
 - -       asin - Inner Span of Channel (cm)
 - -       wgap - Wide Gap Thickness (cm)
 - -       ngap - Narrow Gap Thickness (cm)
 - -    cradius - Inner Radius of Channel Corner (cm)
 - -       fsrd - Clad Surface to Clad Surface Separation (cm)
 - -      cfsrd - Clad Surface to Inner Channel Surface
 - -              Separation (cm)
 - -     rpitch - Fuel Rod Pitch (cm)
 - -        cod - Cladding Outer Diameter (cm)
 - -        cld - Cladding Thickness (cm)
 - -        pod - Fuel Pellet Outer Diameter (cm)
 - - fp_density - Density Vector for Lattice
*/
  float cthick, asin, wgap, ngap, cradius, fsrd, cfsrd, rpitch
   ,cod, cld, pod;
  float *fp_density;
/* - Character Datasets
 - -  matdsnam - name for material dataset
 - -    frcmat - Material Identifier for Fuel Rod Cladding
 - -     fcmat - Material Identifier for Channel
 - -       gds - name for lattice geometry dataset
*/
  char frcmat[6], fcmat[6];
  ascii_string matdsnam, gds;
/* - Structured Variable(s)
 - -  ptr_fg - pointer to proper fuel material dataset
 - -   fmids - pointer to fuel material dataset
*/
  fg_list *ptr_fg = ptr_lg_ds;
  ll_material *fmids;
/* - Adjust Character String Read with FORTRAN for Processing in C -  */
```

```
  { short int n;
    int len = 132, length;
    ascii_string *p = lmdsnam;
    for( n = 1; n < nlat; n++,p++) continue;
    strncpy(matdsnam,*p,132);
    length = mchar(&len,matdsnam);
    matdsnam[length] = '\0';
  }
/* - Find Corresponding Lattice Geometry Dataset */
  { short int n;
    int *p = ptr_correspondence_table;
/*    do{
      p++;p++;}
    while(*p != nlat);
*/
    while(*p != nlat) {p++;p++;}
    nglat = *(p+1);
    for( n = 0; n < (nglat-1); n++) ptr_fg++;
    latdim = ptr_fg->latdim;
    strcpy(gds,ptr_fg->gds_name);
  }
/* - Allocate Memory for Lattice Map */
  lattice = memory_integer(1,(latdim*latdim),lattice);
/* - Load Material Dataset for Lattice - - - - - - - - - - - - - - - - - - */
  fmids = load_fuel_material(&nft,fprefix,matdsnam,latdim,&lattice
   ,&fp_density);
/* - Create Lattice Model - - - - - - - - - - - - - - - - - - - - - - - - - */
   cthick = ptr_fg->cthick;    asin = ptr_fg->asin;
     wgap = ptr_fg->wgap;      ngap = ptr_fg->ngap;
  cradius = ptr_fg->cradius;   fsrd = ptr_fg->fsrd;
    cfsrd = ptr_fg->cfsrd;   rpitch = ptr_fg->rpitch;
      cod = ptr_fg->cod;        cld = ptr_fg->cld;
      pod = ptr_fg->pod;
  strcpy(frcmat,ptr_fg->frcmat);
  strcpy(fcmat,ptr_fg->fcmat);
  if(strstr(gds,"ge7x7") != NULL)
    ge7x7_lattice(cthick,asin,wgap,ngap,cradius,fsrd,cfsrd,rpitch,cod
     ,cld,pod,frcmat,fcmat,latdim,nft,lattice,ncell,nmaterial,lu8,lu9
     ,lu10,ptr_surface_usage,ptr_material_usage,inchannel_density
     ,bypass_density,nuniverse,fp_density,matdsnam,fmids,ptr_core_mtls
     ,nlat,ufl,nsurface,inchannel_material);
   else
    ge8x8_lattice_swr(cthick,asin,wgap,ngap,cradius,fsrd,cfsrd,rpitch,cod
     ,cld,pod,frcmat,fcmat,latdim,nft,lattice,ncell,nmaterial,lu8,lu9
     ,lu10,ptr_surface_usage,ptr_material_usage,inchannel_density
     ,bypass_density,nuniverse,fp_density,matdsnam,fmids,ptr_core_mtls
     ,nlat,ufl,nsurface,inchannel_material);
/* - Return Memory */
  { short int n;
    ll_material *p = fmids, *p_last;
    for( n = 0; n < (nft-1); n++,p++) continue;
    p_last = p;
    for( n = nft; n > 0; n--){
      p = p_last;
```

```
      p_last--;
      rollup_llm(p);}
  }
  lattice = memory_integer(-1,(latdim*latdim),lattice);
}
```

Function material_match

```
#include<stdio.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                            struct ascii_record *last;
                            ascii_string line;
                            struct ascii_record *next;
                           } a_record;
typedef struct s_material{
                          struct s_material *last;
                          int atomic_number;
                          int mass_number;
                          float weight_percentage;
                          char library_suffix[5];
                          struct s_material *next;
                         } ll_material;

ll_material *memory_s_material(int,int,ll_material *);


ll_material *material_match(a_record *p,char name[],float *density
 ,int *n_entries){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * m a t e r i a l _ m a t c h *  Matches Material Identifiers
 - -                                   with Materials on Linked List
 - -                                   for Core Materials
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -          p - pointer to first element in linked list        (input)
 - -       name - string containing name of material sought      (input)
 - -    density - density of material                            (output)
 - -  n_entries - number of elements/isotopes of which the        (output)
 - -              material is composed
 - -    m_first - pointer to first element in material defin-    (output)
 - -              ition link list
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -           n - counter for processing elements within material
 - -               definition
 - -  n_elements - number of elements of which a material is composed
 - -  n_isotopes - number of isotopes for a particular element in a
 - -               material
 - -    a_number - atomic number from material in core materials
 - -               linked list
 - -    m_number - mass number of isotope
```

```
*/
  short int n;
  int n_elements, n_isotopes, a_number, m_number;
/* - Float Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -   ewp - elemental weight-percentage from core materials linked
 - -         list
 - -   iwp - isotopic weight-percentage from core materials linked
 - -         list
*/
  float ewp, iwp;
/* - Character Variable(s)
 - -  db_name - string to hold name of material from linked list
 - -     line - buffer to hold "line" from linked list
 - - l_suffix - library suffix from element or isotope on core mater-
 - -            ial linked list
 - -  element - elemental name from element on core material linked
 - -            list
*/
  char db_name[6], l_suffix[5], element[15];
  ascii_string line;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structure Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  pm_current - pointer to current position in linked list for
 - -               material definition
 - -     pm_next - pointer to next position in linked list for material
 - -               definition
 - -     m_first - first element in linked list to material definition
*/
  ll_material *pm_current, *pm_next, *m_first = NULL;
/* - Sweep through the linked list seeking material identifier match */
  while(p->next != 0){
    strcpy(line,p->line);
    if(strstr(line,name) != NULL){          /* - match on identifier */
      *n_entries = 0;
      sscanf(line,"%s %f %i",db_name,density,&n_elements);
      for(n = 1; n <= n_elements; n++){     /* - sweep through ele - */
        p = p->next;                         /* - ment definitions -  */
        sscanf(p->line,"%s %f %i %s %i"
         ,element,&ewp,&a_number,l_suffix,&n_isotopes);
        pm_next = memory_s_material(1,1,pm_next);
        if(m_first == NULL) {
          m_first = pm_next;
          pm_current = pm_next;
          pm_current->last = NULL;}
        else{
          pm_current->next = pm_next;
          pm_next->last = pm_current;
          pm_current = pm_next;
        }
        pm_current->next = NULL;
        if(strlen(l_suffix) == 1){
```

```
          { short int i;
            *n_entries += n_isotopes;
            for(i = 1;i <= n_isotopes;i++){
              p = p->next;
              sscanf(p->line,"%i %f %s",&m_number,&iwp,l_suffix);
              iwp = ewp*iwp/100.0;
              pm_current->atomic_number = a_number;
              pm_current->mass_number = m_number;
              pm_current->weight_percentage = iwp;
              strcpy(pm_current->library_suffix,l_suffix);
              if(i < n_isotopes){
                pm_next = memory_s_material(1,1,pm_next);
                pm_current->next = pm_next;
                pm_next->last = pm_current;
                pm_current = pm_next;}
            }
          }}
        else{
          (*n_entries)++;
          pm_current->atomic_number = a_number;
          pm_current->mass_number = 0;
          pm_current->weight_percentage = ewp;
          strcpy(pm_current->library_suffix,l_suffix);
          { short int i;
            for(i = 1;i <= n_isotopes; i++) p=p->next;}
        }
      }
      return m_first;
    }
    p = p->next;
  }
/* - No match found -- fatal error - - - - - - - - - - - - - - - - - - - */
  lines(4);
  fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
  fprintf(nout," material match:\n");
  fprintf(nout,"0No Match Found for Input Material %s\n",name);
  abort();
}
```

Function search_fau_list

```
int search_fau_list(int index,int position,int *fau_fill,int nu_cc){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * s e a r c h _ f a u _ l i s t  *  searches list of fuel assembly
 - -                                     assignments to control cells
 - -                                     and returns fuel assembly in-
 - -                                     dex at desired location
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      index - control cell sequence number for which the      (input)
 - -              contents are sought
 - -   position - location of fuel assembly                       (input)
 - -              (1 - NW, 2 - NE, 3 - SE, 4 - SW)
 - -   fau_fill - array providing fuel assembly universe index    (input)
 - -              assignments for each unique control cell
```

```
 - -      nu_cc - starting number for control cell universe      (input)
 - -              indices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -     n - loop variable
 - -     k - loop variable
 - - value - return value
*/
  short int n,k;
  int value;
/* - Sweep through fuel assembly index assignments seeking target control
 - - cell index */
/*  for(n = 1;n < ((index-nu_cc-1)-1);n++) */
  for(n = 1;n <= (index-nu_cc);n++)
    for(k = 1;k <= 4;k++) fau_fill++;
/* - Get fuel assembly index from proper position */
  for(n = 1;n < position;n++) fau_fill++;
  value = *fau_fill;
  return value;
}
```

Function source_specification

```
#include <stdio.h>

void source_specification(int nsrck,float rkk,int ikz,int kct,FILE *lu10
 ,int ncolcc,int nrowcc,int *ccmap,int *ccmapw,float apitch,float afl
 ,int core_f,int ndm,int mct,int ndmp,int ncell,float tempk){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  s o u r c e _ s p e c i f i c a t i o n  *  Adds Source Specif-
 - -                                                ication to Input
 - -                                                Deck
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      nsrck - nominal source size per cycle in MCNP          (input)
 - -        rkk - initial guess for eigenvalue in MCNP           (input)
 - -        ikz - number of cycles to be skipped before begin-   (input)
 - -              ning tally accumulation in MCNP
 - -        kct - number of cycles to be performed in MCNP       (input)
 - -       lu10 - stream pointer for material scratch file       (input)
 - -     ncolcc - Number of Columns in Control Cell Map          (input)
 - -     nrowcc - Number of Rows in Control Cell Map             (input)
 - -      ccmap - Map of Control Cell Universe Number            (input)
 - -     ccmapw - Map Containing Contents of Each Control Cell   (input)
 - -     apitch - fuel assembly pitch (cm)                       (input)
 - -        afl - active fuel length (cm)                        (input)
 - -     core_f - fraction of core modeled in problem            (input)
 - -              (0 - full,
 - -               1 - half,
 - -               4 - quarter)
 - -        ndm - periodicity of dumps to TPE file               (input)
 - -        mct - flag to creation of MCTAL file                 (input)
 - -       ndmp - maximum number of dumps on file                (input)
```

```
 - -      ncell - total number of cells in the problem          (input)
 - -      tempk - problem temperature (Kelvin)                  (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        i - loop variable for columns in map
 - -        j - loop variable for rows in map
 - -        n - utility loop variable
 - -     nsrc - number of source fields
 - -  ptr_map - pointer to ccmap array
 - - ptr_mapw - pointer to ccmapw array
 - -   i_zero - column for origin of problem
 - -   j_zero - row of origin of problem
*/
  short int i,j,n,i_zero,j_zero;
  int nsrc = 0;
  int *ptr_map = ccmap, *ptr_mapw = ccmapw;
/* - Float Variable(s)
 - -   x - abscissa for center of source body
 - -   y - ordinate for center of source body
*/
  float x,y;
/* - Print Program Control and Source Cards - - - - - - - - - - - - - - - */
/* - The thermal scattering kernal temperature in MeV is computed from*/
/* - the formula given on page 3-108 of the MCNP-4B User's Manual - - */
  fprintf(lu10,"c     Control Cards\n");
  fprintf(lu10,"c\n");
  fprintf(lu10," tmp1 %10.4E %ir\n",(8.617E-11*tempk),(ncell-1));
  fprintf(lu10,"kcode  %5i %5.3f %3i %5i\n",nsrck,rkk,ikz,kct);
  fprintf(lu10,"print\n");
  fprintf(lu10,"prdmp %i %i %i %i\n",kct,ndm,mct,ndmp);
/* - Source Fields */
  fprintf(lu10," sdef  erg=d1 pos=d2 axs= 0 0 1 rad=d3 ext=d4\n");
  fprintf(lu10,"  sp1  -3\n");
  fprintf(lu10,"  si2  L");
  for( j = 1; j <= nrowcc; j++){
    for( i = 1; i <= ncolcc; i++,ptr_map++){
/* - Check for Null Control Cell */
      if(*ptr_map == 0){
        for(n = 1;n <= 6; n++) ptr_mapw++;
        continue;}
/* - Compute Coordinates of Source Body Center */
      switch (core_f){
      case 4:
        i_zero = ncolcc;
        j_zero = 1;
        break;
      case 1:
        i_zero = ncolcc/2;
        j_zero = nrowcc/2;
        break;
      case 2:
        i_zero = ncolcc;
```

```
        j_zero = nrowcc/2;
        break;
      }
      if((i == i_zero) || (j == j_zero)) continue;
      nsrc++;
      x = (i-i_zero)*2*apitch;
      y = (j-j_zero)*2*apitch;
      if(nsrc == 1)
        fprintf(lu10," %10.4E %10.4E %10.4E\n",x,y,(afl/2));
      else
        fprintf(lu10,"         %10.4E %10.4E %10.4E\n",x,y
         ,(afl/2));
    }
  }
  fprintf(lu10,"  sp2");
  { int nmax=10,nmin=1;
    do{
      for(n = nmin;n <= nmax;n++)
        fprintf(lu10," 1");
      fprintf(lu10,"\n");
      if(n <= nsrc) fprintf(lu10,"     ");
      nmin += 10;
      nmax += 10;
      if(nmax > nsrc) nmax = nsrc;
    }while(n <= nsrc);
  }
  fprintf(lu10,"  si3 %10.4E\n",((2*apitch)-0.01));
  fprintf(lu10,"  si4 %10.4E\n",(afl/2));
}
```

Function spacer_location

```
void spacer_location(int naxial,int nbundlg,float afl
 ,int *ptr_n_spacer,float *ptr_spacer_location,int *ptr_spacer_node){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  s p a c e r _ l o c a t i o n  *  Determines Nodal Locations
 - -                                       of each Spacer
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     naxial - number of axial nodes                          (input)
 - -    nbundlg - number of unique fuel assembly geometries      (input)
 - -        afl - active fuel length (cm)                        (input)
 - - ptr_n_spacer
 - -            - number of spacers in each fuel assembly type (input)
 - - ptr_spacer_location
 - -            - pointer to array containing location of spa- (input)
 - -              cers for each fuel assembly type (cm)
 - - ptr_spacer_node
 - -            - pointer to array containing space node      (output)
 - -              locations for each fuel assembly type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -       i - utility index variable
```

```
 - -       j - utility index variable
 - -       k - utility index variable
 - -   ptr_n - pointer to integer vector
 - -  ptr_sn - pointer to integer vector containing spacer nodal
 - -           locations
*/
  short int i,j,k;
  int *ptr_n = ptr_n_spacer, *ptr_sn = ptr_spacer_node;
/* - Float Variable(s)
 - - -      ptr_f - pointer to floating point array
 - - node_height - node height assuming uniformly high nodes
 - -     ref_loc - temporary storage for node interface location
*/
  register float ref_loc;
  float node_height;
  float *ptr_f = ptr_spacer_location;
/* - Sweep over the Unique Fuel Assembly Geometry Types - - - - - -  */
  node_height = afl/((float) naxial);
  for(i = 1;i <= nbundlg;i++){
/* - Loop over the Spacers in this Geometry Type */
    for(j = 1;j <= *ptr_n;j++){
      if(*ptr_f <= node_height){
        *ptr_sn = 1;
        ptr_sn++;
        ptr_f++;
        continue;}
      if(*ptr_f > (afl - node_height)){
        *ptr_sn = naxial;
        ptr_sn++;
        ptr_f++;
        continue;}
      ref_loc = node_height;
      for(k = 2;k < naxial;k++){
        if((*ptr_f > ref_loc) &&
         (*ptr_f <= (ref_loc+node_height))){
          *ptr_sn = k;
          ptr_sn++;
          ptr_f++;
          break;}
        else
          ref_loc += node_height;
      }
    }
  }
}
```

Function vessel_generation

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
```

```
                                struct ascii_record *next;
                               } a_record;
typedef struct s_material{
                              struct s_material *last;
                              int atomic_number;
                              int mass_number;
                              float weight_percentage;
                              char library_suffix[5];
                              struct s_material *next;
                             } ll_material;

typedef struct u_list{
                         struct u_list *last;
                         int index;
                         ascii_string label;
                         struct u_list *next;
                        } usage_list;

typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                        } surface_usage_list;


ll_material *material_match(a_record *,char[],float * ,int *);
ll_material *memory_s_material(int, ll_material *);
void rollup_llm(ll_material *);
usage_list *memory_usage_list(int,usage_list *);
surface_usage_list *memory_surface_usage_list(int
 ,surface_usage_list *);
usage_list *load_usage_list(char label[],int index,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
int mchar(int *,char[]);
void header();
void lines(int);
void bufferpad(char[],int,int);
void abort();
void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_symmetry_surfaces(int,FILE *,FILE *,surface_usage_list *
 ,int);
void add_material(FILE *,int *,char[],int,ll_material *);

void vessel_generation(float apitch,float vod,float vthick,float sod
```

```
 ,float sthick,float tutpr,float tcgr,float bltpr,float bcpr
 ,FILE *lu8, FILE *lu9, FILE *lu10,int *ncell,int *nsurface
 ,int *nmaterial,a_record *ptr_core_mtls,char mvessel[],char mshroud[]
 ,char mtg[],char mcp[],char mutp[],char mltp[]
 ,int core_f,surface_usage_list *ptr_surface_usage
 ,usage_list **ptr_material_usage,float bypass_density,float afl){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * v e s s e l _ g e n e r a t i o n *  Generates Vessel Compo-
 - -                                          nents in MCNP Deck
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     apitch - fuel assembly pitch                              (input)
 - -        sod - shroud outer radius                              (input)
 - -     sthick - shroud thickness                                 (input)
 - -        vod - pressure vessel outer radius                     (input)
 - -     vthick - pressure vessel thickness                        (input)
 - -      tutpr - Top of Upper Tie Plate Region                    (input)
 - -       tcgr - Top of Core Grid Region                          (input)
 - -      bltpr - Bottom of Lower Tie Plate Region                 (input)
 - -       bcpr - Bottom of Core Plate Region                      (input)
 - - incore_loc - positions for incore instrumentation             (input)
 - -        lu8 - pointer to scratch file for cell representa-     (input)
 - -              tations
 - -        lu9 - pointer to scratch file for surface repre-       (input)
 - -              tations
 - -       lu10 - pointer to scratch file for material repre-      (input)
 - -              sentations
 - -      ncell - number of MCNP gometrical cells in the in-      (output)
 - -              put card images thus far
 - -   nsurface - number of MCNP gometrical surfaces in the       (output)
 - -              input card images thus far
 - -  nmaterial - number of MNCP materials in the problem         (output)
 - -              thus far
 - -  ptr_core_mtls
 - -            - pointer to linked list of core materials        (input)
 - -    mvessel - material identifier for vessel                  (input)
 - -    mshroud - material identifier for core shroud             (input)
 - -        mtg - material identifier for top grid region         (input)
 - -        mcp - material identifier for core plate region       (input)
 - -       mutp - material identifier for upper tie plate         (input)
 - -              region
 - -       mltp - material identifier for lower tie plate         (input)
 - -              region
 - -     core_f - fraction of core in problem                      (input)
 - -              (1 - full, 2 - half, 4 - quarter)
 - -  ptr_surface_usage
 - -            - pointer to linked list for surface labels      (output)
 - -              and indices
 - -  ptr_material_usage
 - -            - pointer to linked list for material labels     (output)
 - -              and indices
 - -  bypass_density
 - -            - density for bypass water in problem (g/cc)      (input)
 - -        afl - active fuel length (cm)                         (input)
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -          ns - counter for number of "virtual" surfaces accmulated
 - -   n_entries - number of elements and isotopes in a particular
 - -               material
 - -       index - index used for searching usage lists
 - -        zero - zero value;
 - -         pid - process identifier of job creating input deck
 - -     version - version number for linkage code
*/
  short int ns;
  int n_entries,index,zero = 0;
  extern short version;
  extern long pid;
/* - Float Variable(s)
 - -  density - material density
*/
  float density;
/* - Character Variable(s)
 - -     buffer - string variable used to load entire lines for print
 - -      label - label for editing
 - -    ptr_buf - pointer to buffer;
 - -       zaid - MCNP element/isotope identifier
 - -   mnemonic - surface type mnemonic
 - - case_title - title for job
 - -      value - definition of surface
 - -     codemn - name of linkage code
 - -      cdate - date of linkage code execution
 - -     crtime - time of linkage code execution
 - - modification_level
 - -            - modification level for linkage code
*/
  char zaid[11], mnemonic[4];
  extern char case_title[];
  ascii_string buffer, label, value;
  char *ptr_buf;
  extern char codenm[5];
  extern char cdate[9];
  extern char crtime[9];
  extern char modification_level;
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition link list
 - -       pm - working pointer to material definition link list
 - -   ptr_sl - pointer to current record in surface usage link list
 - -   ptr_ml - pointer to current record in material usage link list
*/
  ll_material *ptr_mtl, *pm;
  usage_list *ptr_ml = NULL;
```

```
  surface_usage_list *ptr_sl = NULL;
/* - Write Title for Case and Header Records for Cell Representations */
  fprintf(lu8,case_title);
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Input Deck Generated by %s, Version %2i:%c\n"
   ,codenm,version,modification_level);
  fprintf(lu8,"c     Generated on %s at %s by Process %i\n"
   ,cdate,crtime,pid);
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Cell Cards\n");
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Cells Defining Problem Domain\n");
  fprintf(lu8,"c\n");
/* - Write Header Records for Surface Representations - - - - - - - - */
  fprintf(lu9,"c     Surface Cards\n");
  fprintf(lu9,"c\n");
  fprintf(lu9,"c     Surfaces for Problem Domain\n");
/* - Write Header Records for Material Representations - - - - - - - - */
  fprintf(lu10,"c     Material Cards\n");
/* - Write Header for Deck Generation Edit Table */
  header();
  lines(5);
  fprintf(nout
   ,"OMCNP Input Card Representation Generation Summary Table\n");
  fprintf(nout
   ,"0 Cell Surface(s)     Material/Index/Density    Universe");
  fprintf(nout," Definition\n");
  fprintf(nout,"\n");
/* - Vessel - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell Cards */
  (*nmaterial)++;
  { int len = 6, length;
    length = mchar(&len,mvessel);
    mvessel[length] = '\0';
  }
  ptr_mtl = material_match(ptr_core_mtls,mvessel,&density
   ,&n_entries);
  *ptr_material_usage = load_usage_list(mvessel,1,*ptr_material_usage);
  ptr_ml = *ptr_material_usage;
  strcpy(label,"Pressure Vessel");
  add_cell(label,mvessel,lu8,ncell,*nmaterial,-density,&zero);
/* - Surface Cards */
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  strcpy(mnemonic,"pz");
  index = -1;
  sprintf(value,"%10.4E",(tcgr+30.0));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  strcpy(mnemonic,"pz");
  index = 1;
  sprintf(value,"%10.4E",(bcpr-30.0));
```

```
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Maximum Radial Extent of Problem */
  strcpy(label,"Maximum Radial Extent of Problem");
  strcpy(mnemonic,"cz");
  index = -1;
  sprintf(value,"%10.4E",(vod/2));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Boundaries, if Present */
  if((core_f == 2) || (core_f == 4)){
    (*nsurface)++;
    strcpy(label,"X-Z Plane");
    strcpy(mnemonic,"py");
    strcpy(value,"0.0");
    ptr_sl = ptr_surface_usage;
    while(ptr_sl->next != NULL) ptr_sl = ptr_sl->next;
    ptr_sl = load_surface_usage_list(label,*nsurface,value
     ,mnemonic,"",ptr_sl);
    fprintf(lu8," %i",*nsurface);
    fprintf(lu9,"c      %s\n",ptr_sl->label);
    fprintf(lu9,"    *%1i %s 0.0\n",*nsurface,mnemonic);
    sprintf(buffer,"        %6i",*nsurface);
    bufferpad(buffer,strlen(buffer),59);
    ptr_buf = buffer+59*sizeof(char);
    sprintf(ptr_buf,"%s\n",label);
    lines(1);
    fprintf(nout,buffer);
    if(core_f == 4){
      (*nsurface)++;
      strcpy(label,"Y-Z Plane");
      strcpy(mnemonic,"px");
      strcpy(value,"0.0");
      ptr_sl = load_surface_usage_list(label,*nsurface,value
       ,mnemonic,"",ptr_sl);
      fprintf(lu8," %i",-(*nsurface));
      fprintf(lu9,"c      %s\n",ptr_sl->label);
      fprintf(lu9,"    *%1i %s 0.0\n",*nsurface,mnemonic);}
      sprintf(buffer,"        %6i %s",-(*nsurface),mnemonic);
      bufferpad(buffer,strlen(buffer),59);
      ptr_buf = buffer+59*sizeof(char);
      sprintf(ptr_buf,"%s\n",label);
      lines(1);
      fprintf(nout,buffer);
  }
/* - Inner Radial Surface of Vessel */
  strcpy(label,"Inner Radial Surface of Vessel");
  strcpy(mnemonic,"cz");
  index = 1;
  sprintf(value,"%10.4E",((vod/2)-vthick));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n= 1.0\n");
/* - Material Cards */
```

```
  add_material(lu10,nmaterial,mvessel,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - "Outside" World - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  strcpy(label,"""Outside"" World");
  add_cell(label,"void",lu8,ncell,0,0.0,&zero);
/* - Edit List of Surfaces */
  fprintf(lu8," (");
/* - Maximum Radial Extent of Problem */
  index = 3;
  search_surface_usage_list(0,label,&index,"","","""
   ,ptr_surface_usage);
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Problem */
  index = 1;
  search_surface_usage_list(0,label,&index,"","","""
   ,ptr_surface_usage);
  fprintf(lu8,":");
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  index = 2;
  search_surface_usage_list(0,label,&index,"","","""
   ,ptr_surface_usage);
  fprintf(lu8,":");
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4)){
    strcpy(label,"X-Z Plane");
    search_surface_usage_list(1,label,&index,"","","""
     ,ptr_surface_usage);
    fprintf(lu8,":");
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);
    if(core_f == 4){
      strcpy(label,"Y-Z Plane");
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      fprintf(lu8,":");
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
  }
  fprintf(lu8,") ");
  fprintf(lu8,"\n      imp:n= 0.0\n");
/* - Jet Pump Region - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  (*nmaterial)++;
  strcpy(label,"Jet Pump Region");
  add_cell(label,"Bypass Water",lu8,ncell,*nmaterial,-bypass_density
```

```
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Vessel */
  strcpy(label,"Inner Radial Surface of Vessel");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Outer Radial Surface of Core Shroud */
  strcpy(label,"Outer Radial Surface of Core Shroud");
  strcpy(mnemonic,"cz");
  index = 1;
  sprintf(value,"%10.4E",(sod/2));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
```

```
/*  - Material(s) */
  { short int i, nloc;
    char *cp;
    strcpy(label,"Bypass Water");
    fprintf(lu10,"c      %s\n",label);
    ptr_ml = load_usage_list(label,*nmaterial,ptr_ml);
    strcpy(zaid,"001001.50c");
    nloc = 0;
    if(*nmaterial < 1000) nloc++;
    if(*nmaterial < 100) nloc++;
    if(*nmaterial < 10) nloc++;
    strcpy(label,"2.0");
    { short int ip;
      for(ip = 1;ip <= nloc;ip++)
        fprintf(lu10," ");
        fprintf(lu10,"m%i %s %s\n"
         ,*nmaterial,zaid,label);
    }
    strcpy(zaid,"008016.50c");
    strcpy(label,"1.0");
    fprintf(lu10,"       %s %s\n"
      ,zaid,label);
    { short int ip;
      for(ip = 1;ip <= (nloc-1);ip++) fprintf(lu10," ");
      fprintf(lu10,"mt%i lwtr.01\n",*nmaterial);
    }
  }
/* - Core Shroud - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  strcpy(label,"Core Shroud");
/* - Load Core Materials Database */
  { int len = 6, length;
    length = mchar(&len,mshroud);
    mshroud[length] = '\0';
  }
  search_usage_list(1,mshroud,&index,*ptr_material_usage);
  if(index == 0){
    (*nmaterial)++;
    index = *nmaterial;
    ptr_mtl = material_match(ptr_core_mtls,mshroud,&density
     ,&n_entries);
    ptr_ml = *ptr_material_usage;
    while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
    index = (ptr_ml->index)+1;
    ptr_ml = load_usage_list(mshroud,index,ptr_ml);}
   else
     n_entries = 0;
  add_cell(label,mshroud,lu8,ncell,index,-density,&zero);
/* - Surface(s) */
/* - Outer Radial Surface of Core Shroud */
  strcpy(label,"Outer Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  strcpy(mnemonic,"cz");
  sprintf(value,"%10.4E",((sod/2)-sthick));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/*  - Material(s) */
  if(n_entries != 0){
    add_material(lu10,nmaterial,mshroud,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
  }
/* - Upper Tie Plate Region - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
```

```
    length = mchar(&len,mutp);
    mutp[length] = '\0';
  }
  strcpy(label,mutp);
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mutp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mutp,index,ptr_ml);
  strcpy(label,"Upper Tie Plate Region");
  add_cell(label,mutp,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Active Fuel */
  strcpy(label,"Top of Active Fuel");
  strcpy(mnemonic,"pz");
  index = 1;
  sprintf(value,"%10.4E",afl);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Top of Upper Tie Plate Region */
  strcpy(label,"Top of Upper Tie Plate Region");
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",tutpr);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mutp,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Core Grid Region - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mtg);
    mtg[length] = '\0';
```

```
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mtg,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mtg,index,ptr_ml);
  strcpy(label,"Core Grid Region");
  add_cell(label,mtg,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Upper Tie Plate Region */
  strcpy(label,"Top of Upper Tie Plate Region");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Core Grid Region */
  strcpy(label,"Top of Core Grid Region");
  strcpy(mnemonic,"pz");
  index = -1;
  sprintf(value,"%10.4E",tcgr);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mtg,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Upper Plenum Region - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
```

```
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,*ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Material not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Upper Plenum Region");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Core Grid Region */
  strcpy(label,"Top of Core Grid Region");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
```

```
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
/* - Bypass Water Assumed in this Region */
/* - Lower Tie Plate Region - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mltp);
    mltp[length] = '\0';
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mltp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mltp,index,ptr_ml);
  strcpy(label,"Lower Tie Plate Region");
  add_cell(label,mltp,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Active Fuel */
  strcpy(label,"Bottom of Active Fuel");
  strcpy(mnemonic,"pz");
  sprintf(value,"0.0");
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Bottom of Lower Tie Plate Region */
  strcpy(label,"Bottom of Lower Tie Plate Region");
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",bltpr);
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
```

```
/* - Material(s) */
  add_material(lu10,nmaterial,mltp,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Fuel Support/Core Plate Region - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mcp);
    mcp[length] = '\0';
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mcp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mcp,index,ptr_ml);
  strcpy(label,"Fuel Support/Core Plate");
  add_cell(label,mcp,lu8,ncell,index,-density,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Lower Tie Plate Region */
  strcpy(label,"Bottom of Lower Tie Plate Region");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Fuel Support/Core Grid Region */
  strcpy(label,"Bottom of Fuel Support/Core Plate Region");
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",bcpr);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
```

```
    ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mcp,n_entries,ptr_mtl);
/* - Lower Plenum Region - - - - - - - - - - - - - - - - - - - - - - -  */
/* - Cells */
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,*ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Material not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  sprintf(label,"Lower Plenum Region");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Fuel Support/Core Plate Region */
  strcpy(label,"Bottom of Fuel Support/Core Plate Region");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
```

```
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
/* - Bypass Water Assumed in this Region */
}
```

CONTENTS

		Page
1.	Introduction	3
2.	Routines for IDSGEN, Version 1	4
2.1.	Main Routine	4
2.2.	Service Routines	9
2.3.	Input Routines	16
2.4.	Routines for Processing Unexposed Fuel	26
2.5.	Routines for Processing Exposed Fuel	38

TABLES

Page

1-1 Location of Listings for Each Functional Block 3

1. Introduction

This attachment contains listings of the FORTRAN routines that comprise version 1 of the IDSGEN code. The listings are divided by functional block as in the description of the code given in Attachment VIII and Table 1-1 gives the sections in which the listing for each functional block are located.

Table 1-1 Location of Listings for Each Functional Block

Section	Contents
2.1	Main Function
2.2	Service Routines
2.3	Input Routines
2.4	Routines Processing Unexposed Fuel Lattices
2.5	Routines Processing Exposed Fuel Lattices

2. Routines for IDSGEN, Version 1

2.1. Main Routine

```
      PROGRAM IDSGEN(FPREFIX)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  I D S G E N  *  Creates Intermediate Fuel Material Composition
C - -                     Datasets for Use in Automated Linkage between
C - -                     SAS2H and MCNP
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Modification Log:
C - -  Version  Mod  Description
C - -  -------  ---  -----------------------------------------------
C - -     0      -   Initial Release
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Command Line Variable(s)
C - -   FPREFIX - prefix for input and output files
C - -
C - - Logical Units used for Code - - - - - - - - - - - - - - - - - - - -
C - -      LU3 - (3) Database Containing Nuclide List used to Create
C - -                Dataset for Exposed Fuel
C - -      NIN - (5) Input Unit
C - -     NOUT - (6) Output Unit
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Parameter Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
C - -   MXMEM - maximum number of storage locations available in blank
C - -          common buffer
      PARAMETER (MXMEM=20000)
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  OPTION - Processing Option
C - -          (0 - Unexposed Fuel, 1 - Exposed Fuel)
C - -   FADEX - Fuel Assembly Index Number
C - -   SMEAR - Flag for Unexposed Fuel Cases to Indicate that the
C - -          Average Results should be Uniformly Applied to all the
C - -          Fuel Rods
C - -          (0 - Do Not Smear, 1 - Smear, 2- Smear between gad and nogad
only)
C - -    MDEX - Single Character Indicating Fuel Manufacturer
C - -     FIN - Name for Input File
C - -    FOUT - Name for Output File
C - -  INNAME - Name for NAMELIST Directive
C - -  NDNAME - Name for SAS2H Cut File
C - -   DBTTL - Title on Database File
C - - DB_NAME - File Specification for Database File
C - -     CUT - Flag to Indicate whether the SAS2H Output is a Full
C - -          Output or a "Cut" File
C - -          (0 - Full Output, 1 - "Cut" File)
C - -   NOGAD - Flag to Indicate whether Gadolinia present as an
C - -          Integral Burnable Absorber should be included in the
C - -          Fuel Material Intermediate Dataset
C - -          (0 - Delete Gadolinia, 1 - Retain Gadolinia)
      INTEGER OPTION,FADEX,SMEAR,CUT,NOGAD
```

```
      CHARACTER*1 MDEX
      CHARACTER*7 INNAME
      CHARACTER*80 NDNAME,FIN,FOUT,FPREFIX,DBTTL
      CHARACTER*132 DB_NAME
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
      COMMON /FILES/ LU1,LU2,LU3
      COMMON /MMANAG/ NEXT,MXAVIL,MXUSED
C - - Blank Common - - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON A(MXMEM)
C - - Data Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      DATA INNAME/'$FUEL'/
C - - Initializations - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MXAVIL - Maxmimum Number of Storage Locations Available in the
C - -           Code (Set by a PARAMETER Statement in the Main Routine)
      MXAVIL = MXMEM
C - - Construct Names for Input and Output Files - - - - - - - - - - -
      WRITE(FIN,'(A)') FPREFIX
      WRITE(FIN((MCHAR(80,FIN)+1):),'('''.inp''')')
      WRITE(FOUT,'(A)') FPREFIX
      WRITE(FOUT((MCHAR(80,FOUT)+1):),'('''.out''')')
C - - Open Input and Output Files
      CALL FOPEN(NIN,FIN)
      CALL FOPEN(NOUT,FOUT)
      CALL INHEAD
C - - Echo Input File
      CALL ECHO(INNAME)
C - - Read Input - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  L1 - DENSITY,  L2 - PELOD  ,  L3 - ENRICH ,  L4 - GDCON
C - -  L5 - MAP
      CALL READIN(OPTION,NLAT,MDEX,FADEX,NODE,NDNAME,A
     2 ,NTYPE,L1,L2,L3,L4,L5,SMEAR,DB_NAME,HEIGHT,CUT,NOGAD,NOFPGD)
        CALL EDITIN(OPTION,NLAT,MDEX,FADEX,NODE,NDNAME
     2 ,NTYPE,A(L1),A(L2),A(L3),A(L4),A(L5),SMEAR,DB_NAME,HEIGHT
     3 ,CUT,NOGAD,NOFPGD)
C - - Covert Height to cm
      HEIGHT = 2.54*HEIGHT
C - - Read Database of Valid Nuclides and Associated MCNP Identifiers -
C - -  L26 - NMEM,   L27 - MPRFX  , L28 - MSUFX
      CALL FOPEN(LU3,DB_NAME)
      CALL READDB(LU3,A,NNUCLD,L26,L27,L28,DBTTL)
      CALL FCLOSE(LU3)
      CALL EDITDB(NNUCLD,DBTTL,A(L26),A(L27),A(L28))
      IF(OPTION .EQ. 0) THEN
C - - Compute Number of Fuel Rods of Each Type
C - -  L6 - NRODS
        L6 = MEMORY(NTYPE,1)
        CALL CNRODS(NTYPE,NLAT,A(L5),A(L6))
C - - Compute Average Parameters
C - -   L7 - MUO2  ,  L8 - MU    ,  L9 - MO    , L10 - MUI
C - -  L11 - MGD2O3, L12 - AMUO2 , L13 - UAW   , L14 - W24
C - -  L15 - W26   , L16 - W28   , L17 - AMU   , L18 - AMO
C - -  L19 - AMGD2O3,L20 - FUI   , L21 - FGDI  , L22 - AMGD
C - -  L23 - MGD   , L24 - MGDI  , L25 - RTFI
```

```
      L25 = MEMORY(12*NTYPE,1)
      L20 = MEMORY(4,1)
      L21 = MEMORY(7,1)
      L7 = MEMORY(NTYPE,1)
      L8 = MEMORY(NTYPE,1)
      L9 = MEMORY(NTYPE,1)
      L10 = MEMORY(4*NTYPE,1)
      L11 = MEMORY(NTYPE,1)
      L12 = MEMORY(NTYPE,1)
      L13 = MEMORY(NTYPE,1)
      L14 = MEMORY(NTYPE,1)
      L15 = MEMORY(NTYPE,1)
      L16 = MEMORY(NTYPE,1)
      L17 = MEMORY(NTYPE,1)
      L18 = MEMORY(NTYPE,1)
      L19 = MEMORY(NTYPE,1)
      L22 = MEMORY(NTYPE,1)
      L23 = MEMORY(NTYPE,1)
      L24 = MEMORY(7*NTYPE,1)
      CALL AVERAG(SMEAR,NTYPE,A(L6),A(L1),A(L2),A(L3),A(L4)
     2  ,A(L7),A(L8),A(L9),A(L10),A(L11),A(L12),A(L13),A(L14),A(L15)
     3  ,A(L16),A(L17),A(L18),A(L19),A(L20),A(L21),FO,A(L22),A(L23)
     4  ,A(L24),A(L25))
      CALL EDTAVG(SMEAR,NTYPE,A(L6),A(L1),A(L2),A(L3),A(L4),A(L7)
     2 ,A(L8),A(L9),A(L10),A(L11),A(L12),A(L13),A(L14),A(L15),A(L16)
     3 ,A(L17),A(L18),A(L19),A(L20),A(L21),FO,A(L22),A(L23),A(L24)
     4 ,A(L25))
      L24 = MEMORY(7*NTYPE,-1)
      L23 = MEMORY(NTYPE,-1)
      L22 = MEMORY(NTYPE,-1)
      L19 = MEMORY(NTYPE,-1)
      L18 = MEMORY(NTYPE,-1)
      L17 = MEMORY(NTYPE,-1)
      L16 = MEMORY(NTYPE,-1)
      L15 = MEMORY(NTYPE,-1)
      L14 = MEMORY(NTYPE,-1)
      L13 = MEMORY(NTYPE,-1)
      L12 = MEMORY(NTYPE,-1)
      L11 = MEMORY(NTYPE,-1)
      L10 = MEMORY(4*NTYPE,-1)
      L9 = MEMORY(NTYPE,-1)
      L8 = MEMORY(NTYPE,-1)
      L7 = MEMORY(NTYPE,-1)
C - - Create Dataset for Unexposed Fuel
      CALL DSGEN(SMEAR,MDEX,NLAT,NTYPE,A(L6),A(L1),A(L2),A(L3),A(L4)
     2  ,A(L5),A(L20),A(L21),FO,A(L25),FADEX,NNUCLD,A(L27),A(L28)
     3  ,NODE)
     ELSE
C - - Exposed Fuel - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  L29 - INM   , L30 = WPI
      L29 = MEMORY(NNUCLD,1)
      L30 = MEMORY(NNUCLD,1)
      CALL FOPEN(LU2,NDNAME)
      IF(CUT .EQ. 1) THEN
```

```
          CALL SLOAD(LU2,NNUCLD,A(L29),A(L26),NOGAD,NOFPGD)
        ELSE
          CALL SLOAD2(LU2,NNUCLD,A(L29),A(L26),NOGAD)
        ENDIF
        CALL FCLOSE(LU2)
C - - Compute UO2 Mass for Unexposed Fuel, Average Enrichment and Aver-
C - - age Density
C - -  L31 - NRODS
        L31 = MEMORY(NTYPE,1)
        CALL CMASS(NLAT,NTYPE,A(L1),A(L2),A(L3),A(L5),HEIGHT,UO2NM
     2    ,EAVG,RHOAVG,A(L4),GDCAVG,A(L31),FNVOL)
C - - Generate Oxygen Mass for SAS2H Data
        CALL MOGEN(NNUCLD,A(L29),UO2NM,EAVG,A(L26),GDCAVG)
C - - Edit Nuclide Nodal Masses, Total Mass and Average Enrichment and
C - - Compute Weight Percentages
        CALL MASSES(NNUCLD,UO2NM,EAVG,RHOAVG,A(L29),A(L26),A(L27)
     2    ,A(L28),A(L30),FNVOL,RHOEFF)
C
C - - Option on smear for exposed fuel
C - - Write Dataset for Exposed Fuel, Smeared isotopics
       IF (SMEAR .EQ. 1) THEN
         CALL DSGENE(MDEX,NLAT,NTYPE,A(L31),RHOEFF,A(L2),A(L5),FADEX
     2     ,NNUCLD,A(L27),A(L28),EAVG,GDCAVG,A(L30),NODE)
      ELSE
C - - Write Dataset for Exposed Fuel, discrete gd isotopics
         CALL DSGEND(SMEAR,MDEX,NLAT,NTYPE,A(L31),RHOEFF,A(L2)
     2     ,A(L3),A(L4),A(L5),FADEX,NNUCLD,A(L27),A(L28)
     3     ,EAVG,GDCAVG,A(L30),NODE)
       ENDIF
      ENDIF
      CALL FCLOSE(LU2)
C - - End of Processing- - - - - - - - - - - - - - - - - - - - - - - - - - -
      CALL MEMSUM
      CALL FCLOSE(NOUT)
      END
      BLOCK DATA
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  B L O C K  D A T A  * Initialization of Names Common Blocks
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /LINECM/ NLINE,MAXLIN
      COMMON /PAGES/ NPAGE,NIN,NOUT
      COMMON /CODEID/ CODENM(6),TITLE,VERSON,MOD,PID
      CHARACTER*1 CODENM,MOD
      CHARACTER*2 VERSON
      CHARACTER*5 PID
      CHARACTER*122 TITLE
      COMMON /FILES/ LU1,LU2,LU3
      COMMON /AWEIGHT/ AWU(4),AWGD(7),AGD(7),AWO
C - - Common Block /LINECM/ - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MAXLIN - Maximum Number of Lines on a Page
      DATA MAXLIN/62/
C - - Common Block /PAGES/ - - - - - - - - - - - - - - - - - - - - - - - -
```

```
C - -     NIN - Logical Unit Number for Input to Code
C - -    NOUT - Logical Unit Number of Output from Code
      DATA NIN /5/, NOUT /6/
C - - Common Block /CODEID/ - - - - - - - - - - - - - - - - - - - - - - -
C - -  CODENM - Name of Program
C - -   TITLE - Title for Case
C - -  VERSON - Code Version
C - -     MOD - Code Modification Index
      DATA CODENM /'I','D','S','G','E','N'/
      DATA TITLE /' '/
      DATA VERSON /' 0'/
      DATA MOD /'-'/
C - - Common Block /FILES/ - - - - - - - - - - - - - - - - - - - - - - - -
C - -     LU1 - Logical Unit Number for Fuel Material Database File
C - -     LU2 - Logical Unit Number for SAS2H Cut File
C - -     LU3 - Logical Unit Number for Valid Nuclide List
      DATA LU1/1/,LU2/2/,LU3/3/
C - - Common Block /AWEIGHT/ - - - - - - - - - - - - - - - - - - - - - -
C - -  AWU - Atomic Weights for Uranium Isotopes
C - - AWGD - Atomic Weights for Gadolinia Isotopes
C - -  AGD - Natural Abundances for Gadolinia Isotopes (Atom Fraction
C - -        as a Percentage)
C - -  AWO - Atomic Weight for Oxygen
C - - (N.b., values are from Nuclides and Isotopes, 15th Edition,
C - -  General Electric Co. and KAPL Inc., 1996.)
      DATA AWU/234.040945,235.043922,236.045561,238.050785/
      DATA AWGD/151.919789,153.920862,154.922619,155.922120
     2 ,156.923957,157.924101,159.92051/
      DATA AGD/0.20,2.18,14.8,20.47,15.65,24.84,21.86/
      DATA AWO/15.9994/
      END
```

2.2. Service Routines

Subroutine abort

```
      SUBROUTINE ABORT
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  A B O R T  * Handles abnormal terminations detected by the
C - -                   calling routine
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      WRITE(NOUT,6000)
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - - -
      STOP
C - - Format statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
 6000 FORMAT('0***  Processing stops  ***')
      END
```

Subroutine fclose

```
       SUBROUTINE FCLOSE(UNIT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  F C L O S E  *  Closes Sequential Files
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -        UNIT - Logical Unit Number Associated with the     (input)
C - -               File to be Closed
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER UNIT
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      CLOSE(UNIT=UNIT,STATUS='KEEP',IOSTAT=IOS,ERR=500)
C - - Normal End of Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 WRITE(NOUT,7000) IOS,UNIT
      CALL ABORT
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - - Error format(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. FCLOSE',/
     2 ,' -- Error Number ',I4,' Encountered Closing Logical Unit:',I2)
      END
```

Subroutine fopen

```
      SUBROUTINE FOPEN(UNIT,FNAME)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  F O P E N  *  Opens Sequential Files for Processing
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -        UNIT - Logical Unit Number to Associate with the   (input)
C - -               File to be Opened
C - -       FNAME - Name of File to Open                        (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER UNIT
      CHARACTER*(*) FNAME
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      OPEN(UNIT=UNIT,NAME=FNAME,IOSTAT=IOS,ERR=500)
C - - Normal End of Processing - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - -
  500 CALL LINES(5)
      WRITE(NOUT,7000) IOS,FNAME
      CALL ABORT
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
C - - Error format(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. FOPEN -- Error '
     2 'Number ',I4,' Encountered Opening File:',/,1X,A)
      END
```

Subroutine header

```
      SUBROUTINE HEADER
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  H E A D E R  * Prints the header at the top of each page
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER*4 GETPID
C - - Common block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /LINECM/ NLINE,MAXLIN
      COMMON /PAGES/ NPAGE,NIN,NOUT
      COMMON /CODEID/ CODENM(6),TITLE,VERSON,MOD,PID
      CHARACTER*1 CODENM,MOD
      CHARACTER*2 VERSON
      CHARACTER*5 PID
      CHARACTER*122 TITLE
      COMMON /RUNINFO/ CTIME,CDATE
      CHARACTER*8 CTIME
      CHARACTER*9 CDATE
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      WRITE(NOUT,6000) CODENM,VERSON,MOD,CDATE,CTIME,PID,NPAGE
     2  ,TITLE
      NLINE = 4
      NPAGE = NPAGE+1
      GO TO 10
C - - Entry point for initialization
      ENTRY INHEAD
C - - Obtain the date and time of execution as well as the serial
C - - number
      CALL DATE(CDATE)
      CALL TIME(CTIME)
      IPID = GETPID()
      PID=' '
      NPAGE = 1
      WRITE(PID,'(I5)') IPID
      WRITE(NOUT,6001) CODENM,VERSON,MOD,CDATE,CTIME,PID,NPAGE
      NPAGE = 2
      NLINE = 4
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - -
   10 RETURN
C - - FORMAT statement(s) - - - - - - - - - - - - - - - - - - - - - - -
C - - Write format(s)
 6000 FORMAT('1*** ',6(A1,1X),' ***  VERS. ',A2,':',A1
     2  ,' ** CRWMS M&O'
     3  ,1X,'DATE: ',A9,', TIME: ',A8,1X,' PID: ',A5
     4  ,T123,' PAGE:',I3,/,1X,A,/)
 6001 FORMAT('1*** ',6(A1,1X),' ***  VERS. ',A2,':',A1
     2  ,' ** CRWMS M&O'
     3  ,1X,'DATE: ',A9,', TIME: ',A8,1X,' PID: ',A5
     4  ,T123,' PAGE:',I3,/,/)
      END
```

Subroutine invalr

```
      SUBROUTINE INVALR(NVAL,VECTOR,VALUE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  I N V A L R *  Initializes a real vector to a single value
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -        NVAL - Number of entries in vector                (input)
C - -      VECTOR - Vector to be initialized                  (output)
C - -       VALUE - Value to initialize vector with            (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION VECTOR(NVAL)
C - - Sweep through vector initializing entries - - - - - - - - - - - -
      DO N = 1,NVAL
        VECTOR(N) = VALUE
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine lines

```
      SUBROUTINE LINES(NL)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  L I N E S  * Computes the number of cumulative lines printed
C - -                  and determines if a page eject is necessary
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -          NL - number of new lines to be printed            (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Common block(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /LINECM/ NLINE,MAXLIN
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      NLINE = NLINE+NL
      IF(NLINE .GT. MAXLIN) THEN
        CALL HEADER
        NLINE = NLINE+NL
      ENDIF
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Function mchar

```
      FUNCTION MCHAR(MAXLEN,BUFFER)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  M C H A R  * Determines the number of non-blank characters in
C - -                  character variable of total length MAXLEN
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -      MAXLEN - dimension of character variable              (input)
C - -      BUFFER - character variable                           (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      CHARACTER*1 BLANK
      CHARACTER*(*) BUFFER
C - - Data statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      DATA BLANK/' '/
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      MCHAR = MAXLEN
      DO 10 NC = MAXLEN,1,-1
      IF(BUFFER(NC:NC) .NE. BLANK) THEN
        MCHAR = NC
        GO TO 20
      ENDIF
   10 CONTINUE
      MCHAR = 0
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - -
   20 CONTINUE
      END
```

Function memory

```
      FUNCTION MEMORY(LENGTH,ALLOC8)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * M E M O R Y *  Manages the allocation of storage in the
C - -                    blank common by returning the next avail-
C - -                    able location.
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -       LENGTH - Number of memory locations requested        (input)
C - -       ALLOC8 - Integer flag for requesting more memory     (input)
C - -                 1 : request additional memory
C - -                -1 : return memory to pool
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER ALLOC8
C - - Blank common - - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON A(1)
C - - Common block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /MMANAG/ NEXT,MXAVIL,MXUSED
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - Determine option for memory allocation  - - - - - - - - - - - - - -
      IF(ALLOC8 .EQ. 1) THEN          ! More storage requested
        NTASKD = NEXT+LENGTH-1
        IF(NTASKD .GT. MXAVIL) GO TO 500
        MEMORY = NEXT
        DO 10 NDEX = NEXT,NTASKD      ! Clear storage locations
   10   A(NDEX) = 0.0D0
        NEXT = NEXT+LENGTH
        IF(NTASKD .GT. MXUSED) MXUSED = NTASKD
      ELSEIF(ALLOC8 .EQ. -1) THEN     ! Return storage
        NEXT = NEXT-LENGTH
        MEMORY = 0
      ELSE
        GO TO 510
      ENDIF
      GO TO 600
C - - Error processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 WRITE(NOUT,7000) NTASKD
      STOP
  510 WRITE(NOUT,7010) ALLOC8
      STOP
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - - -
  600 RETURN
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Error format(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. MEMORY ',/
     2 ,' -- too much memory requested',I8,/
     3 ,' blank common size must be increased')
 7010 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. MEMORY ',/
     2 ,'-- illegal allocation option',/
     3 ,' value encountered was: ',I3)
      END
```

Subroutine memsum

```
      SUBROUTINE MEMSUM
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  M E M S U M  *  Edits Dynamic Memory Usage Statistics
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /MMANAG/ NEXT,MXAVIL,MXUSED
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - Compute Percentage Utilization - - - - - - - - - - - - - - - - - -
      UTILPCT = 100*FLOAT(MXUSED)/FLOAT(MXAVIL)
C - - Edits Statistics
      CALL LINES(5)
      WRITE(NOUT,6000) MXAVIL,MXUSED,UTILPCT
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - - Write format(s)
 6000 FORMAT('0Dynamic Memory Usage Statistics',/
     2 ,'0Limit  Used  Utilization (%)',/
     3 ,1X,I5,1X,I5,6X,F6.2)
      END
```

2.3. Input Routines

Subroutine echo

```
      SUBROUTINE ECHO(INNAME)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  E C H O  *  Copies the input file to the output stream
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -       INNAME - Name for Input NAMELIST                    (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  INFIND - Logical Flag to Indicate that the NAMELIST Directive
C - -           has been Encountered
      LOGICAL INFIND
      CHARACTER*(*) INNAME
      CHARACTER*132 BUFFER
C - - Common block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
      COMMON /CODEID/ CODENM(6),TITLE,VERSON,MOD,PID
      CHARACTER*1 CODENM,MOD
      CHARACTER*2 VERSON
      CHARACTER*5 PID
      CHARACTER*122 TITLE
C - - Data Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      DATA INFIND /.FALSE./
C - - Echo input stream - - - - - - - - - - - - - - - - - - - - - - - - -
      CALL LINES(3)
      WRITE(NOUT,6000)
C - - Read Title Line
      READ(NIN,'(A)',END=20) TITLE
   10 READ(NIN,'(A)',END=20) BUFFER
      IF(BUFFER(1:MCHAR(7,INNAME)) .EQ. INNAME) INFIND = .TRUE.
      CALL LINES(1)
      WRITE(NOUT,'(1X,A)') BUFFER(1:MCHAR(132,BUFFER))
      GO TO 10
C - - Consistency Processing - - - - - - - - - - - - - - - - - - - - - -
   20 IF(.NOT. INFIND) GOTO 500
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      REWIND NIN
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 CALL LINES(4)
      WRITE(NOUT,7000)
      CALL ABORT
      RETURN
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - - Write format(s)
 6000 FORMAT('0Input to IDSGEN:',/)
C - - Error Format(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. ECHO -- '
     2 ,'Input Namelist not Encoutered on Input File')
      END
```

Suboutine editin

```
SUBROUTINE EDITIN(OPTION,NLAT,MDEX,FADEX,NODE,NDNAME
     2 ,NTYPE,DENSITY,PELOD,ENRICH,GDCON,MAP,SMEAR,DB_NAME,HEIGHT
     3 ,CUT,NOGAD,NOFPGD)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  E D I T I N  *  Edits Input Information Read from Input File
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -      OPTION - flag to indicate whether unexposed or      (input)
C - -               exposed fuel dataset is to be created
C - -               (0 - unexposed,
C - -               1 - exposed)
C - -        NLAT - Lattice Dimensionality                     (input)
C - -        MDEX - Single-character Identifier for Fuel       (input)
C - -               Manfacturer
C - -               (G - GE, S - Siemens (or predecessor),
C - -                A - ABB)
C - -       FADEX - Fuel Assembly Identifier                   (input)
C - -        NODE - Axial Node                                 (input)
C - -      NDNAME - Specification for SAS2H "*.cut" File for   (input)
C - -               Exposed Fuel
C - -       NTYPE - Number of Distinct Types of Fuel Rods      (input)
C - -     DENSITY - Stack Density for Each Fuel Rod Type       (input)
C - -       PELOD - Pellet Density for Each Fuel Rod Type      (input)
C - -      ENRICH - Enrichment for Each Fuel Rod Type          (input)
C - -       GDCON - Gadolinia Concentration for Each Fuel Rod  (input)
C - -               Rod Type
C - -       SMEAR - Flag for Unexposed Fuel Cases to Indicate  (input)
C - -               that the Average Results should be
C - -               Uniformly Applied to all the Fuel Rods
C - -               (0 - Do Not Smear, 1 - Smear)
C - -     DB_NAME - File Specification for File Containing     (input)
C - -               the Valid Nuclide List
C - -      HEIGHT - Height of Node (cm)                        (input)
C - -         CUT - Flag to Indicate whether the SAS2H Output  (input)
C - -               is from a Full Output File or from a
C - -               "Cut" File
C - -               (0 - Full Output, 1 - "Cut" File)
C - -       NOGAD - Flag to Indicate whether Gadolinia Incor-  (input)
C - -               porated as a Integral Burnable Absorber
C - -               should be Incorporated in the Fuel Mater-
C - -               ial Intermediate Dataset
C - -               (0 - No, 1 - Yes)
C - -      NOFPGD - Flag to Indicate whether Fission Product   (input)
C - -               Gadolinium Isotopes are to be Incorporated
C - -               in the Fuel Material Intermediate Dataset
C - -               (0 - No, 1 - Yes)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER OPTION,FADEX,SMEAR,CUT,NOGAD
      CHARACTER*80 NDNAME
      CHARACTER*(*) DB_NAME
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - -
      DIMENSION MAP(NLAT,NLAT)
      DIMENSION DENSITY(NTYPE),PELOD(NTYPE),ENRICH(NTYPE)
     2,GDCON(NTYPE)
C - - Edit Contents of NAMELIST Input - - - - - - - - - - - - - - -
      CALL HEADER
      CALL LINES(5)
      WRITE(NOUT,6000)
      CALL LINES(5)
      WRITE(NOUT,6010) OPTION
      WRITE(NOUT,6020) NLAT
      WRITE(NOUT,6030) MDEX
      WRITE(NOUT,6031) DB_NAME
      IF(FADEX .NE. 0) THEN
        CALL LINES(2)
        WRITE(NOUT,6040) FADEX
        WRITE(NOUT,6050) NODE
      ENDIF
      IF(OPTION .EQ. 1) THEN
        CALL LINES(5)
        WRITE(NOUT,6060) NDNAME
        WRITE(NOUT,6061) HEIGHT
        WRITE(NOUT,6062) CUT
        WRITE(NOUT,6063) NOGAD
        WRITE(NOUT,6064) NOFPGD
      ENDIF
      CALL LINES(1)
      WRITE(NOUT,6069) SMEAR
      CALL LINES(1)
      WRITE(NOUT,6070) NTYPE
      CALL LINES(NTYPE+1)
      WRITE(NOUT,6080) (DENSITY(N),N,N=1,NTYPE)
      CALL LINES(NTYPE+1)
      WRITE(NOUT,6090) (PELOD(N),N,N=1,NTYPE)
      CALL LINES(NTYPE+1)
      WRITE(NOUT,6100) (ENRICH(N),N,N=1,NTYPE)
      CALL LINES(NTYPE+1)
      WRITE(NOUT,6110) (GDCON(N),N,N=1,NTYPE)
      CALL LINES(NLAT+3)
      WRITE(NOUT,6120)
      DO J = 1,NLAT
        WRITE(NOUT,6130) (MAP(I,J),I=1,NLAT)
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - -
      RETURN
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - -
C - - Write FORMAT(s)
 6000 FORMAT('0Input Edit -- Values from Input Deck Card Images',/
     2 ,'0Variable    Value    Definition',/)
 6010 FORMAT('    OPTION',T13,3X,I1,T23,'Processing Option '
     2 ,'(0 - Unexposed Fuel, 1 - Exposed Fuel')
 6020 FORMAT('      NLAT',T13,2X,I2,T23,'Lattice Dimensionality')
 6030 FORMAT('      MDEX',T13,3X,A1,T23,'Manufacturer '
```

```
    2 ,'(G - GE, S - Siemens, A - ABB)')
 6031 FORMAT('   DB_NAME',T23,'Valid Nuclide List Database'
    2 ,/,T13,A)
 6040 FORMAT('     FADEX',T13,1X,I3,T23,'Fuel Assembly Index')
 6050 FORMAT('      NODE',T13,2X,I2,T23,'Axial Node')
 6060 FORMAT('    NDNAME',T23,'File Specification for SAS2H File',/
    2 ,T13,A)
 6061 FORMAT('    HEIGHT',T13,F6.3,T23,'Height of Node')
 6062 FORMAT('       CUT',T13,2X,I2,T23
    2 ,'SAS2H Cut File (0 - No, 1 - Yes)')
 6063 FORMAT('     NOGAD',T13,2X,I2,T23
    2,'Delete Burnable Absorber Gadolinia (0 - No, 1 - Yes)')
 6064 FORMAT('    NOFPGD',T13,2X,I2,T23
    2 ,'Delete Fission Product Gadolinium Isotopes'
    3 ,' (0 - No, 1 - Yes)')
 6069 FORMAT('     SMEAR',T13,3X,I1,T23
    2 ,'Smear Average Values Across Lattice (0 - No, 1 - Yes)')
 6070 FORMAT('     NTYPE',T13,2X,I2,T23
    2 ,'Number of Distinct Fuel Rod Types')
 6080 FORMAT('   DENSITY',T23,'Stack Density',/
    2 ,(T13,F5.2,T23,'(',I<(ALOG10(FLOAT(NTYPE))+1)>,')'))
 6090 FORMAT('     PELOD',T23,'Fuel Pellet Outer Diameter (inches)',/
    2 ,(T13,F5.3,T23,'(',I<(ALOG10(FLOAT(NTYPE))+1)>,')'))
 6100 FORMAT('    ENRICH',T23,'Fuel Rod Enrichment',/
    2 ,(T13,F5.3,T23,'(',I<(ALOG10(FLOAT(NTYPE))+1)>,')'))
 6110 FORMAT('     GDCON',T23,'Gadolinia Concentation',/
    2 ,(T13,F5.3,T23,'(',I<(ALOG10(FLOAT(NTYPE))+1)>,')'))
 6120 FORMAT('0Fuel Rod Type Loading Map',/)
 6130 FORMAT(12(1X,I2))
      END
```

Subroutine r2dmap

```
      SUBROUTINE R2DMAP(NCOL,NROW,MAP,LU)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  R 2 D M A P *  Reads Two-dimensional Map giving Fuel
C - -                     Assembly Locations
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -          NCOL - Number of "Columns" in Core Map            (input)
C - -          NROW - Number of "Rows" in Core Map               (input)
C - -           MAP - Map of Indices for Core Locations         (output)
C - -                 (integer)
C - -            LU - Logical Unit from which to Read Map        (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER MAP(NCOL,NROW)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DO J = 1,NROW
        READ(LU,*,END=500) (MAP(I,J),I=1,NCOL)
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 CALL LINES(5)
      WRITE(NOUT,7000) LU
      CALL ABORT
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - - Error format(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. R2DMAP -- '
     2 'Premature End-of-File Encountered on Logical Unit ',I2)
      END
```

Subroutine rdbval

```
      SUBROUTINE RDBVAL(LU1,NNUCLD,NMEM,MPRFX,MSUFX)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  R D B V A L  *  Reads Database File Values
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -        LU1 - Logical Unit Number for Database File      (input)
C - -     NNUCLD - Number of Nuclides in Valid Nuclide List   (input)
C - -       NMEM - Alphanumeric Representation for Nuclides  (output)
C - -              in SAS2H Output
C - -      MPRFX - MCNP Prefixes for Valid Nuclides          (output)
C - -              Nuclides
C - -      MSUFX - MCNP Suffixes for Valid Nuclides          (output)
C - -              Nuclides
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      CHARACTER*4 MSUFX(NNUCLD)
      CHARACTER*6 NMEM(NNUCLD)
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MPRFX(NNUCLD)
C - - Loop through Values in Database File - - - - - - - - - - - - - -
      DO I = 1,NNUCLD
        READ(LU1,'(I6,2X,A6,2X,A4)') MPRFX(I),NMEM(I),MSUFX(I)
      ENDDO
C - - End of Processing - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine readdb

```
      SUBROUTINE READDB(LU,A,NNUCLD,L26,L27,L28,DBTTL)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  R E A D D B  *  Reads Database File for Valid Nuclide Iden-
C - -                     tifiers
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -          LU - Logical Unit Number for Database File      (input)
C - -           A - Blank Common Buffer for Dynamic Memory       (i&o)
C - -               Allocation
C - -      NNUCLD - Number of Nuclides in Valid Nuclide List  (output)
C - -        NMEM - Memory Index for Alphanumeric Representa- (output)
C - -               tion for Nuclides in SAS2H Output
C - -         L26 - Memory Index for MCNP Prefix for Valid    (output)
C - -               Nuclides
C - -       MSUFX - Memory Index for MCNP Suffix for Valid    (output)
C - -               Nuclides
C - -       DBTTL - Title for Valid Nuclide List              (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      CHARACTER*80 DBTTL
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - -
      DIMENSION A(1)
C - - Read Title Record - - - - - - - - - - - - - - - - - - - - - - - -
      READ(LU,'(A)') DBTTL
C - - Read the Number of Entries in the Nuclide List
      READ(LU,*) NNUCLD
C - - Allocate Memory for Nuclides in List
      L26 = MEMORY(6*NNUCLD,1)
      L27 = MEMORY(NNUCLD,1)
      L28 = MEMORY(4*NNUCLD,1)
      CALL RDBVAL(LU,NNUCLD,A(L26),A(L27),A(L28))
C - - End of Processing - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine readin

```
      SUBROUTINE READIN(OPTION,NLAT,MDEX,FADEX,NODE,NDNAME,A
     2 ,NTYPE,L1,L2,L3,L4,L5,SMEAR,DB_NAME,HEIGHT,CUT,NOGAD,NOFPGD)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  R E A D I N  *  Reads Input File for Unexposed Fuel Dataset
C - -                     Creation
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -      OPTION - flag to indicate whether unexposed or     (output)
C - -               exposed fuel dataset is to be created
C - -               (0 - unexposed,
C - -               1 - exposed)
C - -        NLAT - Lattice Dimensionality                    (output)
C - -        MDEX - Single-character Identifier for Fuel      (output)
```

```
C - -                Manfacturer
C - -                (G - GE, S - Siemens (or predecessor),
C - -                 A - ABB).
C - -       FADEX - Fuel Assembly Identifier                  (output)
C - -        NODE - Axial Node                                (output)
C - -      NDNAME - Specification for SAS2H "*.cut" File for  (output)
C - -               Exposed Fuel
C - -           A - Pointer to Dynamic Memory Array            (input)
C - -       NTYPE - Number of Distinct Types of Fuel Rods     (output)
C - -          L1 - Pointer to Stack Density for Each Fuel    (output)
C - -               Rod Type
C - -          L2 - Pointer to Pellet Diameter for Each Fuel  (output)
C - -               Rod Type
C - -          L3 - Pointer to Enrichment for Each Fuel Rod   (output)
C - -               Type
C - -          L4 - Pointer to Gadolinia Concentration for    (output)
C - -               Each Fuel Rod Rod Type
C - -          L5 - Pointer to Fuel Rod Type Map              (output)
C - -       SMEAR - Flag for Unexposed Fuel Cases to Indicate (output)
C - -               that the Average Results should be
C - -               Uniformly Applied to all the Fuel Rods
C - -               (0 - Do Not Smear, 1 - Smear)
C - -     DB_NAME - Name for Database File Containing Valid   (output)
C - -               Nuclide List
C - -      HEIGHT - Node Height (cm)                           (output)
C - -         CUT - Flag to Indicate whether the SAS2H Output (output)
C - -               is a Full Output File or a "Cut" File
C - -               (0 - Full Output, 1 - "Cut" File
C - -       NOGAD - Flag to Indicate whether Gadolinia Incor- (output)
C - -               porated as an Integral Burnable Absorber
C - -               is to be Incorporated in the Fuel Material
C - -               Intermediate Dataset
C - -               (0 - No, 1 - Yes)
C - -      NOFPGD - Flag to Indicate whether Fission Product  (output)
C - -               Gadolinium Isotopes are to be Incorpor-
C - -               ated in the Fuel Material Intermediate
C - -               Dataset (0 - No, 1 - Yes)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER OPTION,FADEX,SMEAR,CUT,NOGAD
      CHARACTER*1 MDEX
      CHARACTER*80 NDNAME
      CHARACTER*132 DB_NAME
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION A(1)
C - - Namelist Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /FUEL/ OPTION,NLAT,MDEX,FADEX,NODE,NTYPE,NDNAME,SMEAR
     2 ,DB_NAME,HEIGHT,CUT,NOGAD,NOFPGD
C - - Skip Title Record - - - - - - - - - - - - - - - - - - - - - - - - -
      READ(NIN,'(1X)')
C - - Read Namelist Input
```

```
      READ(NIN,FUEL)
C - -  L1 - DENSITY,  L2 - PELOD  ,  L3 - ENRICH ,  L4 - GDCON
C - -  L5 - MAP
      L1 = MEMORY(NTYPE,1)
      L2 = MEMORY(NTYPE,1)
      L3 = MEMORY(NTYPE,1)
      L4 = MEMORY(NTYPE,1)
      L5 = MEMORY(NLAT*NLAT,1)
C - - Read Values for Each Fuel Rod Type - - - - - - - - - - - - - - -
C - - Stack Density
      READ(NIN,'(1X)')
      CALL RRD(NTYPE,A(L1),NIN)
C - - Pellet Diameter
      READ(NIN,'(1X)')
      CALL RRD(NTYPE,A(L2),NIN)
C - - Enrichment
      READ(NIN,'(1X)')
      CALL RRD(NTYPE,A(L3),NIN)
C - - Gadolinia Concentration
      READ(NIN,'(1X)')
      CALL RRD(NTYPE,A(L4),NIN)
C - - Fuel Rod Map
      READ(NIN,'(1X)')
      CALL R2DMAP(NLAT,NLAT,A(L5),NIN)
C - - End of Processing - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FCLOSE(NIN)
      RETURN
      END
```

Subroutine rrd

```
      SUBROUTINE RRD(NVAL,VECT,LU)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  R R D *  Reads a Vector of Real Numbers in Free Format
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -          NVAL - Number of Values in Vector                (input)
C - -          VECT - Values in Vector                         (output)
C - -            LU - Logical Unit from which to Read Map       (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION VECT(NVAL)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      READ(LU,*,END=500) (VECT(N),N=1,NVAL)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 CALL LINES(5)
      WRITE(NOUT,7000) LU
      CALL ABORT
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - - Error format(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. RRD -- '
     2 'Premature End-of-File Encountered on Logical Unit ',I2)
      END
```

2.4. Routines for Processing Unexposed Fuel

Subroutine averag

```
      SUBROUTINE AVERAG(SMEAR,NTYPE,NRODS,DENSITY,PELOD,ENRICH
     2 ,GDCON,MUO2,MU,MO,MUI,MGD2O3,AMUO2,UAW,W24,W26,W28
     3 ,AMU,AMO,AMGD2O3,FUI,FGDI,FO,AMGD,MGD,MGDI,RTFI)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  A V E R A G  *  Computes Average Values from Unexposed
C - -                     Fuel Input Data
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -       SMEAR - Flag for Unexposed Fuel Cases to Indicate  (input)
C - -               that the Average Results should be Uni-
C - -               formly Applied to all the Fuel Rods
C - -               (0 - Do Not Smear, 1 - Smear)
C - -       NTYPE - Number of Distinct Types of Fuel Rods      (input)
C - -       NRODS - Number of Fuel Rods in Each Rod Type       (input)
C - -     DENSITY - Stack Density for Each Fuel Rod Type       (input)
C - -       PELOD - Pellet Density for Each Fuel Rod Type      (input)
C - -      ENRICH - Enrichment for Each Fuel Rod Type          (input)
C - -       GDCON - Gadolinia Concentratio for Each Fuel Rod   (input)
C - -               Rod Type
C - -        NGAD - Number of Rods with Unique Gadolinia      (output)
C - -               Loadings
C - -        MUO2 - UO2 Mass by Fuel Rod Type                 (output)
C - -          MU - Uranium Mass by Fuel Rod Type             (output)
C - -          MO - Oxygen Mass by Fuel Rod Type              (output)
C - -         MUI - Uranium Mass by Isotope for Each Fuel Rod (output)
C - -               Type
C - -      MGD2O3 - Gd2O3 Mass by Fuel Rod Type                (output)
C - -       AMUO2 - UO2 Atomic Mass by Fuel Rod Type           (output)
C - -         UAW - Uranium Atomic Weight by Fuel Rod Type     (output)
C - -         W24 - Weight Percentage of U-234 by Fuel Rod     (output)
C - -               Type
C - -         W26 - Weight Percentage of U-236 by Fuel Rod     (output)
C - -               Type
C - -         W28 - Weight Percentage of U-238 by Fuel Rod     (output)
C - -               Type
C - -         AMU - Uranium Atomic Mass by Fuel Rod Type       (output)
C - -         AMO - Oxygen Atomic Mass by Fuel Rod Type        (output)
C - -     AMGD2O3 - Atomic Mass of Gd2O3 by Fuel Rod Type      (output)
C - -         FUI - Isotopic Uranium Mass Fraction for Entire  (output)
C - -               Lattice or Sub-lattice
C - -        FGDI - Isotopic Gadolinia Mass Fraction for       (output)
C - -               Entire Lattice or Sub-lattice
C - -          FO - Oxygen Mass Fraction for Entire Lattice    (output)
C - -               or Sub-lattice
C - -        AMGD - Gadolinium Atomic Mass by Fuel Rod Type    (output)
C - -         MGD - Mass of Gadolinia by Fuel Rod Type         (output)
C - -        MGDI - Isotopic Gadolinium Mass by Fuel Rod Type  (output)
C - -        RTFI - Fractional Isotopic by Fuel Rod Type       (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
```

```
C - - Parameter Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      PARAMETER (PI=3.1415927)
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER SMEAR
      REAL MUO2(NTYPE),MU(NTYPE),MO(NTYPE),MGD2O3(NTYPE)
     2 ,MGD(NTYPE),MGDI(7,NTYPE)
      REAL MUI(4,NTYPE)
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /AWEIGHT/ AWU(4),AWGD(7),AGD(7),AWO
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION FMASSI(4),GDMASSI(7)
      DIMENSION DENSITY(NTYPE),PELOD(NTYPE),ENRICH(NTYPE)
     2 ,NRODS(NTYPE),GDCON(NTYPE),UAW(NTYPE),W24(NTYPE),W26(NTYPE)
     3 ,W28(NTYPE),AMUO2(NTYPE),AMU(NTYPE),AMO(NTYPE)
     4 ,AMGD(NTYPE),FUI(4),FGDI(7),RTFI(12,NTYPE)
C - - Static Functions - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  WF24 - Computes U-234 Weight Percentage from Correlation with
C - -         U-235 Weight Percentage
C - -  WF26 - Computes U-236 Weight Percentage from Correlation with
C - -         U-235 Weight Percentage
C - -  WF28 - Computes U-238 Weight Percentage from Weight Percen-
C - -         tages for U-234, U-235 and U-236
      WF24(X) = 0.007731*((X)**1.0837)
      WF26(X) = 0.0046*X
      WF28(X24,X25,X26) = 100.0-X24-X25-X26
C - - Convert Pellet Outer Diameter (PELOD) to centimeters - - - - -
      DO N = 1,NTYPE
        PELOD(N) = PELOD(N)*2.54
      ENDDO
C - - Rotate through Fuel Rod Types - - - - - - - - - - - - - - - - - -
      GDAW = 0
      DO I = 1,7
        GDAW = GDAW+AWGD(I)*AGD(I)
      ENDDO
      GDAW = GDAW/100.0
      DO N = 1,NTYPE
C - - Compute Uranium Atomic Weight
        UAW(N) = ENRICH(N)*AWU(2)
        W24(N) = WF24(ENRICH(N))
        UAW(N) = UAW(N)+W24(N)*AWU(1)
        W26(N) = WF26(ENRICH(N))
        UAW(N) = UAW(N)+W26(N)*AWU(3)
        W28(N) = WF28(W24(N),ENRICH(N),W26(N))
        UAW(N) = UAW(N)+W28(N)*AWU(4)
        UAW(N) = UAW(N)/100.0
C - - Total Fuel Linear Mass
        FMASS = NRODS(N)*DENSITY(N)*PI*PELOD(N)*PELOD(N)/4.0
        MGD2O3(N) = GDCON(N)*FMASS/100.0
        MUO2(N) = FMASS-MGD2O3(N)
C - - Linear Atomic and Molecular Masses
        AMGD2O3 = MGD2O3(N)/(2*GDAW+3*AWO)
        AMGD(N) = 2*AMGD2O3
        AMUO2(N) = MUO2(N)/(UAW(N)+2*AWO)
        AMU(N) = AMUO2(N)
```

```
        AMO(N) = 2*AMUO2(N)+3*AMGD2O3
        MO(N) = AMO(N)*AWO
        MGD(N) = AMGD(N)*GDAW
        DO I = 1,7
          MGDI(I,N) = MGD(N)*(AWGD(I)*AGD(I)/(100*GDAW))
        ENDDO
        MU(N) = AMU(N)*UAW(N)
        MUI(1,N) = MU(N)*W24(N)/100.0
        MUI(2,N) = MU(N)*ENRICH(N)/100.0
        MUI(3,N) = MU(N)*W26(N)/100.0
        MUI(4,N) = MU(N)*W28(N)/100.0
C - - Compute Fractional Isotopic by Rod Type
        DO I = 1,4
          RTFI(I,N) = MUI(I,N)/FMASS
        ENDDO
        RTFI(5,N) = MO(N)/FMASS
        DO I = 6,12
          RTFI(I,N) = MGDI((I-5),N)/FMASS
        ENDDO
      ENDDO
C - - Compute Fractions for Entire Lattice or Sub-lattice - - - - - -
      IF(SMEAR .EQ. 1) THEN
        FMASS = 0.0
        OMASS = 0.0
        CALL INVALR(4,FMASSI,0.0)
        CALL INVALR(7,GDMASSI,0.0)
        DO N = 1,NTYPE
          OMASS = OMASS+MO(N)
          FMASS = FMASS+MO(N)
          DO I = 1,4
            FMASSI(I) = FMASSI(I)+MUI(I,N)
            FMASS = FMASS+MUI(I,N)
          ENDDO
          DO I = 1,7
            GDMASSI(I) = GDMASSI(I)+MGDI(I,N)
            FMASS = FMASS+MGDI(I,N)
          ENDDO
        ENDDO
        FO = OMASS/FMASS
        DO I = 1,4
          FUI(I) = FMASSI(I)/FMASS
        ENDDO
        DO I = 1,7
          FGDI(I) = GDMASSI(I)/FMASS
        ENDDO
      ENDIF
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine cnrods

```
      SUBROUTINE CNRODS(NTYPE,NLAT,MAP,NRODS)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  C N R O D S  *  Counts the Number of Each Fuel Rod Type
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -       NTYPE - Number of Fuel Rods Types in Lattice        (input)
C - -        NLAT - Lattice Dimensionality                      (input)
C - -         MAP - Fuel Rod Type Map                           (input)
C - -       NRODS - Number of Each Type of Fuel Rod            (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION NRODS(NTYPE)
      DIMENSION MAP(NLAT,NLAT)
C - - Rotate Over Lattice Map Accmulating Sums - - - - - - - - - - - - -
      DO J = 1,NLAT
        DO I = 1,NLAT
          NRODS(MAP(I,J)) = NRODS(MAP(I,J))+1
        ENDDO
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine dsgen

```
SUBROUTINE DSGEN(SMEAR,MDEX,NLAT,NTYPE,NRODS,DENSITY,PELOD
     2 ,ENRICH,GDCON,MAP,FUI,FGDI,FO,RTFI,FADEX,NNUCLD,MPRFX,MSUFX
     3 ,NODE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  D S G E N  *  Creates Fuel Material Intermediate Datasets
C - -                  for Unexposed Fuel
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -       SMEAR - Flag for Unexposed Fuel Cases to Indicate  (input)
C - -               that the Average Results should be Uni-
C - -               formly Applied to all the Fuel Rods
C - -               (0 - Do Not Smear, 1 - Smear)
C - -        MDEX - Single-character Identifier for Fuel       (input)
C - -               Manufacturer
C - -               (G - GE, S - Siemens (or predecessor)
C - -                A - ABB)
C - -        NLAT - Lattice Dimensionality                     (input)
C - -       NTYPE - Number of Distinct Types of Fuel Rods      (input)
C - -       NRODS - Number of Fuel Rods in Each Rod Type       (input)
C - -     DENSITY - Stack Density for Each Fuel Rod Type       (input)
C - -       PELOD - Fuel Pellet Outer Diameter (cm)            (input)
C - -      ENRICH - Enrichment for Each Fuel Rod Type          (input)
C - -       GDCON - Gadolinia Concentration for Each Fuel Rod  (input)
C - -               Rod Type
C - -         MAP - Fuel Rod Type Map                          (input)
C - -         FUI - Isotopic Uranium Mass Fraction for Entire  (input)
C - -               Lattice or Sub-lattice
```

```
C - -        FGDI - Isotopic Gadolinia Mass Fraction for        (input)
C - -               Entire Lattice or Sub-lattice
C - -          FO - Oxygen Mass Fraction for Entire Lattice     (input)
C - -               or Sub-lattice
C - -        RTFI - Fractional Isotopic by Fuel Rod Type        (input)
C - -       FADEX - Index used for Lattice Identification       (input)
C - -      NNUCLD - Total Number of Entries in Valid Nuclide    (input)
C - -               List
C - -       MPRFX - Prefixes for MCNP Nuclide Identifiers in    (input)
C - -               Valid Nuclide List
C - -       MSFUX - Suffixes for MCNP Nuclide Identifiers in    (input)
C - -               Valid Nuclide List
C - -        NODE - Node Number                                 (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -       UMN - Mass Numbers for Uranium Isotopes
C - -      GDMN - Mass Numbers for Gadolinium Isotopes
C - -     LMASS - Scratch Variable to Accumlate the Total Linear Mass
C - -             for the Assembly
C - -   DSFNAME - File Name for Fuel Material Intermediate Dataset
C - -    BUFFER - Scratch Character String used to Manage Output to
C - -             Dataset
      INTEGER SMEAR,FADEX
      INTEGER UMN(4),GDMN(7)
      REAL LMASS
      CHARACTER*1 MDEX
      CHARACTER*4 MSUFX(NNUCLD)
      CHARACTER*80 DSFNAME,BUFFER
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
      COMMON /FILES/ LU1,LU2,LU3
      COMMON /CODEID/ CODENM(6),TITLE,VERSON,MOD,PID
      CHARACTER*1 CODENM,MOD
      CHARACTER*2 VERSON
      CHARACTER*5 PID
      CHARACTER*122 TITLE
      COMMON /RUNINFO/ CTIME,CDATE
      CHARACTER*8 CTIME
      CHARACTER*9 CDATE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
C - -    UMN - Mass Numbers for Uranium Isotopes
C - -   GDMN - Mass Numbers for Gadolinium Isotopes
      DIMENSION MPRFX(NNUCLD)
      DIMENSION DENSITY(NTYPE),ENRICH(NTYPE),NRODS(NTYPE)
     2 ,PELOD(NTYPE),GDCON(NTYPE),FUI(4),FGDI(7),RTFI(12,NTYPE)
      DIMENSION MAP(NLAT,NLAT)
C - - Data Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      DATA UMN/234,235,236,238/
      DATA GDMN/152,154,155,156,157,158,160/
C - - Compute Volume Weighted Enrichment and Gadolinia Concentation -
      AVGD = 0.0
      AVGE = 0.0
      GADC = 0.0
```

```
      GAREA = 0.0
      LMASS = 0.0
      DO NT = 1,NTYPE
        AVGD = AVGD+DENSITY(NT)*NRODS(NT)*(PELOD(NT)*PELOD(NT))
        GAREA = GAREA+NRODS(NT)*(PELOD(NT)*PELOD(NT))
        AVGE = AVGE+
     2   NRODS(NT)*ENRICH(NT)*DENSITY(NT)*(PELOD(NT)*PELOD(NT))
        GADC =
     2   GADC+NRODS(NT)*GDCON(NT)*DENSITY(NT)*(PELOD(NT)*PELOD(NT))
        LMASS = LMASS+NRODS(NT)*DENSITY(NT)*(PELOD(NT)*PELOD(NT))
      ENDDO
      AVGD = AVGD/GAREA
      AVGE = AVGE/LMASS
      GADC = GADC/LMASS
C - - Construct File Name for Dataset - - - - - - - - - - - - - - - - -
      DSFNAME = ' '
      WRITE(DSFNAME,'(A1,I<(ALOG10(FLOAT(NLAT))+1)>)') MDEX,NLAT
      NZ = 3-(IFIX(ALOG10(100*AVGE))+1)
      IF(NZ .GT. 0) THEN
        DO N = NZ,1,-1
          WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''0'')')
        ENDDO
      ENDIF
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2 ,'(I<(ALOG10(100*AVGE)+1)>)')
     3 NINT(100*AVGE)
      IF(GADC .NE. 0) THEN
        NZ = 3-(IFIX(ALOG10(100*GADC))+1)
      ELSE
        NZ = 2
      ENDIF
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''G'')')
      IF(NZ .GT. 0) THEN
        DO N = NZ,1,-1
          WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''0'')')
        ENDDO
      ENDIF
      IF(GADC .GT. 0) THEN
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2   ,'(I<(ALOG10(100*GADC)+1)>)')
     3   NINT(100*GADC)
      ELSE
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2   ,'(''0'')')
      ENDIF
      IF(SMEAR .EQ. 0) THEN
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''D'')')
      ELSE
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''S'')')
      ENDIF
      IF(FADEX .NE. 0) THEN
        IF(FADEX .LT. 100) WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2   ,'(''0'')')
        IF(FADEX .LT. 10) WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
```

```
     2   ,'(''0'')')
         WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2   ,'(I<IFIX(ALOG10(FLOAT(FADEX))+1)>)') FADEX
         IF(NODE .NE. 0) THEN
           IF(NODE .LT. 10) WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2     ,'(''0'')')
           WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2     ,'(I<IFIX(ALOG10(FLOAT(NODE))+1)>)') NODE
         ENDIF
       ENDIF
       WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''.dat'')')
C - - Open File for Processing
       CALL FOPEN(LU1,DSFNAME)
C - - Write Title to Dataset
       IF(MDEX .EQ. 'G') THEN
         BUFFER = 'GE'
       ELSEIF(MDEX .EQ. 'S') THEN
         BUFFER = 'Siemens'
       ELSE
         BUFFER = 'ABB'
       ENDIF
       WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2  ,'('' '',I<(ALOG10(FLOAT(NLAT))+1)>,''x''
     3  ,I<(ALOG10(FLOAT(NLAT))+1)>)') NLAT,NLAT
       WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'(''/Avg Enrichment '',F4.2)') AVGE
       IF(GADC .NE. 0.0)
     2  WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     3   ,'(''/Avg Gadolinia '',F5.3)') GADC
       IF(SMEAR .EQ. 0) THEN
         WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2   ,'(''/Discrete'')')
       ELSE
         WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2   ,'(''/Smeared'')')
       ENDIF
       WRITE(LU1,'(A)') BUFFER
C - - Write Tracking Information to Dataset
       BUFFER = ''
       WRITE(BUFFER,'(''Generated by '',6A1)')
     2 (CODENM(I),I=1,6)
       WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' '',A2,'':'',A1)') VERSON,MOD
       WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' on '',A9,'' at '',A8)') CDATE,CTIME
       WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' by Process '',A5)') PID
       WRITE(LU1,'(A)') BUFFER
C - - Fuel Rod Type Map
       WRITE(LU1,'(''Fuel Rod Type Map'')')
       IF(SMEAR .EQ. 0) THEN
         DO J = 1,NLAT
           WRITE(LU1,'(11(I2,'' ''))')
     2       (MAP(I,J),I=1,NLAT)
```

```
          ENDDO
        ELSE
          DO J = 1,NLAT
            DO I = 1,NLAT
              IF(MAP(I,J) .GE. 1) MAP(I,J) = 1
            ENDDO
          ENDDO
          DO J = 1,NLAT
            WRITE(LU1,'(11(I2,'' '')')
     2        (MAP(I,J),I=1,NLAT)
          ENDDO
        ENDIF
C - - Density Value(s)
        WRITE(LU1,'(''Density Value(s)'')')
        IF(SMEAR .EQ. 0) THEN
          WRITE(LU1,'(5(F6.3,:,'',''))') (DENSITY(N),N=1,NTYPE)
        ELSE
          WRITE(LU1,'(F6.3)') AVGD
        ENDIF
C - - Material Compositions
        WRITE(LU1,'(''Fuel Material Compositions'')')
        IF(SMEAR .EQ. 0) THEN
C - - Uranium Isotopes
          DO NT = 1,NTYPE
            IF(NTYPE .NE. 1) THEN
              WRITE(LU1,'(''Index '',I3,'' 12'')') NT
            ELSE
              WRITE(LU1,'(''12'')')
            ENDIF
            DO NI = 1,4
              BUFFER = ''
              WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2          ,'(''92,'',A4,'','')')
     3          MSUFX(MATCH((92000+UMN(NI)),NNUCLD,MPRFX))
              WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2          ,'(I3,'','')') UMN(NI)
              WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2          ,'(F7.5)') RTFI(NI,NT)
              WRITE(LU1,'(A)') BUFFER
            ENDDO
C - - Oxygen
            BUFFER = ''
            WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(''8,'',A4,'','')')
     3        MSUFX(MATCH(8016,NNUCLD,MPRFX))
            WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(''16,'')')
            WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(F7.5)') RTFI(5,NT)
            WRITE(LU1,'(A)') BUFFER
C - - Gadolnium
            DO NI = 6,12
              BUFFER = ''
              WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
```

```
     2          ,'(''64,'',A4,'','')')
     3          MSUFX(MATCH((64000+GDMN(NI-5)),NNUCLD,MPRFX))
          WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2          ,'(I3'','')') GDMN(NI-5)
          WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2          ,'(F7.5)') RTFI(NI,NT)
          WRITE(LU1,'(A)') BUFFER
        ENDDO
      ENDDO
    ELSE
C - - Uranium Isotopes
      WRITE(LU1,'(''12'')')
      DO NI = 1,4
        BUFFER = ''
        WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(''92,'',A4,'','')')
     3        MSUFX(MATCH((92000+UMN(NI)),NNUCLD,MPRFX))
        WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(I3,'','')') UMN(NI)
        WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(F7.5)') FUI(NI)
        WRITE(LU1,'(A)') BUFFER
      ENDDO
C - - Oxygen
      BUFFER = ''
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2    ,'(''8,'',A4,'','')')
     3    MSUFX(MATCH(8016,NNUCLD,MPRFX))
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2    ,'(''16,'')')
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(F7.5)') FO
      WRITE(LU1,'(A)') BUFFER
C - - Gadolnium
      DO NI = 6,12
        BUFFER = ''
        WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(''64,'',A4,'','')')
     3        MSUFX(MATCH((64000+GDMN(NI-5)),NNUCLD,MPRFX))
        WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2        ,'(I3'','')') GDMN(NI-5)
        WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2          ,'(F7.5)') FGDI(NI-5)
        WRITE(LU1,'(A)') BUFFER
      ENDDO
    ENDIF
C - - Close File
    CALL FCLOSE(LU1)
C - - Copy File to Output - - - - - - - - - - - - - - - - - - - - - -
    CALL FOPEN(LU1,DSFNAME)
    CALL HEADER
    CALL LINES(3)
    WRITE(NOUT,'(''0Intermediate Dataset: '',A,/)') DSFNAME
 10 READ(LU1,'(A)',END=20) BUFFER
```

```
      CALL LINES(1)
      WRITE(NOUT,'(1X,A)') BUFFER
      GOTO 10
   20 CALL CLOSE(LU1)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
C - - Write Format(s)
      END
```

Subroutine edtavg

```
      SUBROUTINE EDTAVG(SMEAR,NTYPE,NRODS,DENSITY,PELOD,ENRICH
     2 ,GDCON,MUO2,MU,MO,MUI,MGD2O3,AMUO2,UAW,W24,W26,W28
     3 ,AMU,AMO,AMGD2O3,FUI,FGDI,FO,AMGD,MGD,MGDI,RTFI)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  E D T A V G  *  Edits Averages of Fuel Isotopics
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -       SMEAR - Flag for Unexposed Fuel Cases to Indicate  (input)
C - -               that the Average Results should be Uni-
C - -               formly Applied to all the Fuel Rods
C - -               (0 - Do Not Smear, 1 - Smear)
C - -       NTYPE - Number of Distinct Types of Fuel Rods      (input)
C - -       NRODS - Number of Fuel Rods in Each Rod Type       (input)
C - -     DENSITY - Stack Density for Each Fuel Rod Type       (input)
C - -       PELOD - Pellet Density for Each Fuel Rod Type      (input)
C - -      ENRICH - Enrichment for Each Fuel Rod Type          (input)
C - -       GDCON - Gadolinia Concentratio for Each Fuel Rod   (input)
C - -               Rod Type
C - -        MUO2 - UO2 Mass by Fuel Rod Type                  (input)
C - -          MU - Uranium Mass by Fuel Rod Type              (input)
C - -          MO - Oxygen Mass by Fuel Rod Type               (input)
C - -         MUI - Uranium Mass by Isotope for Each Fuel Rod  (input)
C - -               Type
C - -      MGD2O3 - Gd2O3 Mass by Fuel Rod Type                 (input)
C - -       AMUO2 - UO2 Atomic Mass by Fuel Rod Type            (input)
C - -         UAW - Uranium Atomic Weight by Fuel Rod Type      (input)
C - -         W24 - Weight Percentage of U-234 by Fuel Rod      (input)
C - -               Type
C - -         W26 - Weight Percentage of U-236 by Fuel Rod      (input)
C - -               Type
C - -         W28 - Weight Percentage of U-238 by Fuel Rod      (input)
C - -               Type
C - -         AMU - Uranium Atomic Mass by Fuel Rod Type        (input)
C - -         AMO - Oxygen Atomic Mass by Fuel Rod Type         (input)
C - -     AMGD2O3 - Atomic Mass of Gd2O3 by Fuel Rod Type       (input)
C - -         FUI - Isotopic Uranium Mass Fraction for Entire   (input)
C - -               Lattice or Sub-lattice
C - -        FGDI - Isotopic Gadolinia Mass Fraction for        (input)
C - -               Entire Lattice or Sub-lattice
C - -          FO - Oxygen Mass Fraction for Entire Lattice     (input)
C - -               or Sub-lattice
C - -        AMGD - Gadolinium Atomic Mass by Fuel Rod Type     (input)
```

```
C - -        MGD - Mass of Gadolinia by Fuel Rod Type          (input)
C - -       MGDI - Isotopic Gadolinium Mass by Fuel Rod Type   (input)
C - -       RTFI - Fractional Isotopic by Fuel Rod Type        (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER SMEAR
      REAL MUO2(NTYPE),MU(NTYPE),MO(NTYPE),MGD2O3(NTYPE)
     2 ,MGD(NTYPE),MGDI(7,NTYPE)
      REAL MUI(4,NTYPE)
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /AWEIGHT/ AWU(4),AWGD(7),AGD(7),AWO
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - -
      DIMENSION DENSITY(NTYPE),PELOD(NTYPE),ENRICH(NTYPE)
     2 ,NRODS(NTYPE),GDCON(NTYPE),UAW(NTYPE),W24(NTYPE),W26(NTYPE)
     3 ,W28(NTYPE),AMUO2(NTYPE),AMU(NTYPE),AMO(NTYPE),AMGD2O3(NTYPE)
     4 ,AMGD(NTYPE),FUI(4),FGDI(7),RTFI(12,NTYPE)
C - - Edit Computation of Uranium Isotopic Weight Percentages, and
C - - Uranium Metal Atomic Weight
      CALL HEADER
      CALL LINES(4+NTYPE)
      WRITE(NOUT,6000)
      DO N = 1,NTYPE
        WRITE(NOUT,6010) N,W24(N),ENRICH(N),W26(N),W28(N),UAW(N)
      ENDDO
C - - Edit Linear Masses, Linear Molecular Masses, and Linear Atomic
C - - Masses
      CALL LINES(5+NTYPE)
      WRITE(NOUT,6020)
      DO N = 1,NTYPE
        FMASS = MUO2(N)+MGD2O3(N)
        WRITE(NOUT,6030) N,FMASS,MGD2O3(N),MUO2(N),AMGD2O3(N)
     2    ,AMUO2(N),AMU(N),AMO(N),AMGD(N),MU(N),MO(N),MGD(N)
      ENDDO
C - - Edit Linear Isotopic Masses by Fuel Type
      CALL LINES(6+NTYPE)
      WRITE(NOUT,6040)
      DO N = 1,NTYPE
        WRITE(NOUT,6050) N,(MUI(I,N),I=1,4),MO(N),(MGDI(I,N),I=1,7)
      ENDDO
C - - Edit Fractional Isotopic by Fuel Type
      CALL LINES(5+NTYPE)
      WRITE(NOUT,6060)
      DO N = 1,NTYPE
        WRITE(NOUT,6070) N,(RTFI(I,N),I=1,12)
      ENDDO
      IF(SMEAR .EQ. 1) THEN
C - - Edit Fractional Isptopics for Entire Lattice or Sub-lattice
        CALL LINES(6)
        WRITE(NOUT,6080)
        WRITE(NOUT,6090) (FUI(I),I=1,4),FO,(FGDI(J),J=1,7)
      ENDIF
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
```

```
      RETURN
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - - Write Format(s)
 6000 FORMAT('0        /----- Weight Percentages ----/ Atomic Weight'
     4    ,/,' Index   U-234   U-235   U-236   U-238     U-metal',/)
 6010 FORMAT(T4,I2,T8,F7.4,T16,F7.4,T24,F7.4,T32,F7.4,T44,F7.3)
 6020 FORMAT('0        /--- Linear Mass ---//- Molecular Mass -/'
     2       ,'/----- Atomic Mass ----//--- Linear Mass -----/',/
     2       ,'                (g/cm)            (g atoms/cm)     '
     3       ,'       (g atoms/cm)              (g/cm)',/
     2       ,' Index  Total   Gd2O3    UO2    Gd2O3    UO2     '
     3       ,'     U      O       Gd       U      O        Gd  ',/)
 6030 FORMAT(T4,I2,T8,F7.3,T15,F7.4,T23,F7.3,T31,F7.4,T39,F7.4
     2 ,T49,F7.4,T57,F7.4,T65,F7.4,T74,F7.3,T82,F7.4,T90,F7.4)
 6040 FORMAT('0Isotopic Weights by Rod Type',/
     2       ,'0Index  U-234   U-235   U-236   U-238     O   '
     3       ,' 152Gd  154Gd  155Gd  156Gd  157Gd  158Gd  160Gd',/)
 6050 FORMAT(T4,I2,T8,F6.4,T16,F6.3,T24,F6.4,T32,F6.2,T40,F6.3
     2 ,T47,F6.4,T54,F6.4,T61,F6.4,T68,F6.4,T75,F6.4,T82,F6.4
     3 ,T89,F6.4)
 6060 FORMAT('0Fractional Isotopics by Fuel Rod Type',/
     2 ,'0Index     U-234   U-235   U-236   U-238     O     152Gd '
     3 ,'  154Gd   155Gd   156Gd   157Gd   158Gd   160Gd',/)
 6070 FORMAT(T4,I2,3X,12(1X,F7.5))
 6080 FORMAT(
     2 '0Fractional Isotopics for Entire Lattice or Sub-lattice',/
     3 ,'0Index     U-234   U-235   U-236   U-238     O     152Gd '
     4 ,'  154Gd   155Gd   156Gd   157Gd   158Gd   160Gd',/)
 6090 FORMAT(8X,12(1X,F7.5))
      END
```

2.5. Routines for Processing Exposed Fuel

Subroutine cmass

```
      SUBROUTINE CMASS(NLAT,NTYPE,DENSITY,PELOD,ENRICH,MAP,HEIGHT
     2 ,UO2NM,EAVG,RHOAVG,GDCON,GDCAVG,NRODS,FNVOL)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  *  C M A S S  *  Computes the Initial UO2 Mass and Average En-
C - -                    richment for Unexposed Fuel that Corresponds to
C - -                    Exposed Fuel from SAS2H
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  Argument(s):
C - -        NLAT - lattice dimensionality                        (input)
C - -       NTYPE - Number of Fuel Rods Types in Lattice          (input)
C - -     DENSITY - Pellet Densities by Fuel Rod Type for         (input)
C - -               Lattice
C - -       PELOD - Pellet Outer Diameter (cm) by Fuel Rod Type   (input)
C - -      ENRICH - Pellet Enrichment (w/o) by Fuel Rod Type      (input)
C - -         MAP - Lattice Map showing Fuel Rod Type Distribu-   (input)
C - -               tion
C - -      HEIGHT - Node Height (cm)                               (input)
C - -       UO2NM - UO2 Mass in Node (g)                          (output)
C - -        EAVG - Lattice-averaged Enrichment (w/o)             (output)
C - -      RHOAVG - Lattice-averaged Density (g/cc)               (output)
C - -       GDCON - Gadolinia Concentrations by Fuel Rod Type      (input)
C - -      GDCAVG - Lattice-averaged Gadolinia Concentration      (output)
C - -               (w/o)
C - -       NRODS - Number of Fuel Rods for Each Fuel Rod Type    (output)
C - -        FVOL - UO2 Volume in Node (cm**3)                    (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Parameter Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
      PARAMETER (PI = 3.1415926535)
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION NRODS(NTYPE)
      DIMENSION MAP(NLAT,NLAT)
      DIMENSION DENSITY(NTYPE),PELOD(NTYPE),ENRICH(NTYPE),GDCON(NTYPE)
C - - Data Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      DATA NTROD/0/
C - - Initialization - - - - - - - - - - - - - - - - - - - - - - - - - - -
      UO2NM = 0.0
      EAVG = 0.0
      RHOAVG = 0.0
      GDCAVG = 0.0
      FNVOL = 0.0
C - - Convert Pellet Outer Diameters to cm
      PELOD = 2.54*PELOD
C - - Determine the Number of Fuel Rods in Each Fuel Rod Type - - - - -
      DO N = 1,NTYPE
        DO J = 1,NLAT
          DO I = 1,NLAT
            IF(MAP(I,J) .EQ. N) NRODS(N) = NRODS(N)+1
          ENDDO
        ENDDO
```

```
      ENDDO
C - - Loop over the Total Number of Fuel Rod Types - - - - - - - - - - -
      DO N = 1,NTYPE
        NTROD = NTROD+NRODS(N)
        VOL = (NRODS(N)*(PI/4)*HEIGHT*(PELOD(N)**2))
        UO2NM = UO2NM+DENSITY(N)*VOL
        FNVOL = FNVOL+VOL
        EAVG = EAVG+(NRODS(N)*ENRICH(N))
        GDCAVG = GDCAVG+GDCON(N)*DENSITY(N)*VOL
        RHOAVG = RHOAVG+(NRODS(N)*DENSITY(N))
      ENDDO
      EAVG = EAVG/FLOAT(NTROD)
      RHOAVG = RHOAVG/FLOAT(NTROD)
      GDCAVG = GDCAVG/UO2NM
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine dsgend

```
SUBROUTINE DSGEND(SMEAR,MDEX,NLAT,NTYPE,NRODS,RHOEFF,PELOD
     2 ,ENRICH,GDCON,MAP,FADEX,NNUCLD,MPRFX,MSUFX
     3 ,EAVG,GDCAVG,WPI,NODE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  D S G E N  *  Creates Fuel Material Intermediate Datasets
C - -                  for Exposed Fuel with discrete gadolinia placement
C - -                  only in originally gd containing rods
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -       SMEAR - Flag for Unexposed Fuel Cases to Indicate  (input)
C - -               that the Average Results should be Uni-
C - -               formly Applied to all the Fuel Rods
C - -               (0 - Do Not Smear, 1 - Smear all, 2- Smear between gad and
nogad only)
C - -        MDEX - Single-character Identifier for Fuel       (input)
C - -               Manufacturer
C - -               (G - GE, S - Siemens (or predecessor)
C - -                A - ABB)
C - -        NLAT - Lattice Dimensionality                     (input)
C - -       NTYPE - Number of Distinct Types of Fuel Rods      (input)
C - -       NRODS - Number of Fuel Rods in Each Rod Type       (input)
C - -      RHOEFF - Effective Density for Fuel                 (input)
C - -       PELOD - Fuel Pellet Outer Diameter (cm)            (input)
C - -      ENRICH - Enrichment for Each Fuel Rod Type          (input)
C - -       GDCON - Gadolinia Concentration for Each Fuel Rod  (input)
C - -               Rod Type
C - -         MAP - Fuel Rod Type Map, overwritten for SMEAR=2 (input)
C - -       FADEX - Index used for Lattice Identification      (input)
C - -      NNUCLD - Total Number of Entries in Valid Nuclide   (input)
C - -               List
C - -       MPRFX - Prefixes for MCNP Nuclide Identifiers in   (input)
C - -               Valid Nuclide List
C - -       MSFUX - Suffixes for MCNP Nuclide Identifiers in   (input)
C - -               Valid Nuclide List
C - -        EAVG - Lattice-averaged Enrichment (w/o)          (input)
```

```
C - -     GDCAVG - Lattice-averaged Gadolinium Concentration  (input)
C - -              (w/o)
C - -        WPI - Weight Percentages for Nuclides in Exposed (input)
C - -              Fuel (w/o)
C - -       NODE - Node Number                                 (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - -       RTFI - Fractional Isotopic by Fuel Rod Type        (input)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - -      UMN - Mass Numbers for Uranium Isotopes
C - -     GDMN - Mass Numbers for Gadolinium Isotopes
C - -  DSFNAME - File Name for Fuel Material Intermediate Dataset
C - -   BUFFER - Scratch Character String used to Manage Output to
C - -            Dataset
      INTEGER SMEAR,FADEX
      INTEGER UMN(4),GDMN(7)
      CHARACTER*1 MDEX
      CHARACTER*4 MSUFX(NNUCLD)
      CHARACTER*80 DSFNAME,BUFFER
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
      COMMON /FILES/ LU1,LU2,LU3
      COMMON /CODEID/ CODENM(6),TITLE,VERSON,MOD,PID
      CHARACTER*1 CODENM,MOD
      CHARACTER*2 VERSON
      CHARACTER*5 PID
      CHARACTER*122 TITLE
      COMMON /RUNINFO/ CTIME,CDATE
      CHARACTER*8 CTIME
      CHARACTER*9 CDATE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - -
C - -   UMN - Mass Numbers for Uranium Isotopes
C - -  GDMN - Mass Numbers for Gadolinium Isotopes
      DIMENSION MPREFX(NNUCLD)
      DIMENSION DENSITY(NTYPE),ENRICH(NTYPE),NRODS(NTYPE)
     2 ,PELOD(NTYPE),GDCON(NTYPE)
     3 ,RTFI(NNUCLD,NTYPE),WPI(NNUCLD)
      DIMENSION MAP(NLAT,NLAT)
C - - Data Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      DATA UMN/234,235,236,238/
      DATA GDMN/152,154,155,156,157,158,160/
C
C - - 789012345678901234567890123456789012345678901234567890123456789012
C
C - - If NTYPE=1 then don't allow SMEAR=2, as there's no need
      IF (NTYPE .EQ. 1) SMEAR=0
C
C - - Renormalized input weight percentages into rod types
C - - Determine total number of rods, and number of rod types with gd
      NTRODS=0
      NTGD=0
```

```
      DO NT = 1, NTYPE
        NTRODS=NTRODS+NRODS(NT)
        IF (GDCON(NT) .NE. 0) NTGD=NTGD+1
C - -   Set constant density equal to effective density
        DENSITY(NT) = RHOEFF
      ENDDO
C - - For SMEAR=0 Group all rods types discretly
      NTOUT=0
      IF (SMEAR .EQ. 0) THEN
       NTOUT=NTYPE
       DO NT = 1, NTYPE
             DO NN = 1, NNUCLD
               IF (GDCON(NT) .EQ. 0) THEN
C - - Check if gd isotope
                 IF (MPRFX(NN)/1000 .EQ. 64) THEN
                   RTFI(NN,NT) = 0.0
                 ELSE
               RTFI(NN,NT)=WPI(NN)*REAL(NRODS(NT))/(REAL(NTRODS))
             ENDIF
           ELSE
C - - Check if gd isotope
C - - 789012345678901234567890123456789012345678901234567890123456789012
                 IF (MPRFX(NN)/1000 .EQ. 64) THEN
                  RTFI(NN,NT) = WPI(NN)/REAL(NTGD)
                 ELSE
              RTFI(NN,NT)=WPI(NN)*REAL(NRODS(NT))/(REAL(NTRODS))
             ENDIF
           ENDIF
             ENDDO
       ENDDO
      ELSE
       NTOUT=SMEAR
C - - Redo Fuel Rod Type Map for SMEAR=2 option, noGad=1, Gad=2
      WRITE(LU1,'(''Fuel Rod Type Map'')')
      DO J = 1,NLAT
        DO I = 1,NLAT
         IF (MAP(I,J) .GT. 0) THEN
           IF(GDCON(MAP(I,J)) .EQ. 0) THEN
             MAP(I,J)=1
            ELSE
             MAP(I,J)=2
            ENDIF
          ENDIF
        ENDDO
      ENDDO
C - - Group only into gd and non gad rod types
       DO NT = 1, NTOUT
             DO NN = 1, NNUCLD
               IF (NT .EQ. 1) THEN
                 IF (MPRFX(NN)/1000 .EQ. 64 ) THEN
                   RTFI(NN,NT) = 0.0
                  ELSE
                   RTFI(NN,NT)=WPI(NN)*(1.0 - REAL(NTGD)/(REAL(NTYPE)))
                 ENDIF
```

```
                ELSE
C - -           Gd group
                RTFI(NN,NT)=WPI(NN)*REAL(NTGD)/(REAL(NTYPE))
              ENDIF
                ENDDO
         ENDDO
        ENDIF
C
C - - Construct File Name for Dataset - - - - - - - - - - - - - - - - -
        DSFNAME = ' '
        WRITE(DSFNAME,'(A1,I<(ALOG10(FLOAT(NLAT))+1)>)') MDEX,NLAT
        NZ = 3-(IFIX(ALOG10(100*EAVG))+1)
        IF(NZ .GT. 0) THEN
          DO N = NZ,1,-1
            WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''0'')')
          ENDDO
        ENDIF
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
       2 ,'(I<(ALOG10(100*EAVG)+1)>)')
       3 NINT(100*EAVG)
C - - Ensure no roundoff of character width of Gad percentages when near 1%
        IF(GDCAVG .NE. 0) THEN
          IF (GDCAVG .LT. 1.0 .AND. GDCAVG .GT. 0.995) GDCAVG=1.0
          NZ = 3-(IFIX(ALOG10(100*GDCAVG))+1)
        ELSE
          NZ = 2
        ENDIF
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''G'')')
        IF(NZ .GT. 0) THEN
          DO N = NZ,1,-1
            WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''0'')')
          ENDDO
        ENDIF
        IF(GDCAVG .NE. 0) THEN
          WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
       2    ,'(I<(ALOG10(100*GDCAVG)+1)>)')
       3    NINT(100*GDCAVG)
        ELSE
          WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
       2    ,'(''0'')')
        ENDIF
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''D'')')
        IF(FADEX .LT. 100)
       2  WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
       3    ,'(''0'')')
        IF(FADEX .LT. 10)
       2  WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
       3    ,'(''0'')')
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
       2 ,'(I<IFIX(ALOG10(FLOAT(FADEX)))+1>)') FADEX
        IF(NODE .LT. 10)
       2  WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
       3    ,'(''0'')')
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
```

```
     2 ,'(I<IFIX(ALOG10(FLOAT(NODE)))+1>)') NODE
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''.dat'')')
C - - Open File for Processing
      CALL FOPEN(LU1,DSFNAME)
C - - Write Title to Dataset
      IF(MDEX .EQ. 'G') THEN
        BUFFER = 'GE'
      ELSEIF(MDEX .EQ. 'S') THEN
        BUFFER = 'Siemens'
      ELSE
        BUFFER = 'ABB'
      ENDIF
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2  ,'('' '',I<(ALOG10(FLOAT(NLAT))+1)>,''x''
     3  ,I<(ALOG10(FLOAT(NLAT))+1)>)') NLAT,NLAT
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'(''/Avg Enrichment '',F4.2)') EAVG
      IF (GADC .NE. 0.0)
     2  WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     3   ,'(''/Avg Gadolinia '',F5.3)') GDCAVG
C
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2   ,'(''/Discrete'')')
      WRITE(LU1,'(A)') BUFFER
C - - Write Tracking Information to Dataset
      BUFFER = ''
      WRITE(BUFFER,'(''Generated by '',6A1)')
     2 (CODENM(I),I=1,6)
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' '',A2,'':'',A1)') VERSON,MOD
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' on '',A9,'' at '',A8)') CDATE,CTIME
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' by Process '',A5)') PID
      WRITE(LU1,'(A)') BUFFER
C - - Fuel Rod Type Map
      WRITE(LU1,'(''Fuel Rod Type Map'')')
      DO J = 1,NLAT
          WRITE(LU1,'(11(I2,'' ''))')
     2       (MAP(I,J),I=1,NLAT)
      ENDDO
C - - Density Value(s)
      WRITE(LU1,'(''Density Value(s)'')')
      WRITE(LU1,'(5(F6.3,:,'',''))') (DENSITY(N),N=1,NTOUT)
C - - Material Compositions
      WRITE(LU1,'(''Fuel Material Compositions'')')
C - - Uranium Isotopes
C - - Zero out isotopics with 0 mass except for gd, do only for first fuel
type
      NIT = 0
      DO NN = 1,NNUCLD
       IF(RTFI(NN,1) .GT. 0.0 .OR. (MPRFX(NN)/1000) .EQ. 64) NIT=NIT+1
      ENDDO
C
```

```
C - - 789012345678901234567890123456789012345678901234567890123456789012
C
      DO NT = 1,NTOUT
C         Reduce total isotopic count by gd isotopes(7) if non-gd rod
              IF (GDCON(NT) .GT. 0 .OR. (SMEAR .NE. 0 .AND. NT .EQ. 2)) THEN
                  NITGD=NIT
               ELSE
                  NITGD=NIT-7
              ENDIF
          IF (NTYPE .NE. I) THEN
            WRITE(LU1,'(''Index '',I3,1P,I3)') NT, NITGD
           ELSE
            WRITE(LU1,'(I3)') NITGD
          ENDIF
        DO NN = 1, NNUCLD
          IF (RTFI(NN,NT) .GT. 0.0) THEN
           WRITE(LU1,'(I3,'','',A4,'','',I3,'','',1P,E10.4)')
     2      (MPRFX(NN)/1000),MSUFX(NN)
     3      ,(MPRFX(NN)-1000*(MPRFX(NN)/1000)),RTFI(NN,NT)
          ENDIF
        ENDDO
      ENDDO
C
C - - Close File
      CALL FCLOSE(LU1)
C - - Copy File to Output - - - - - - - - - - - - - - - - - - - - - -
      CALL FOPEN(LU1,DSFNAME)
      CALL HEADER
      CALL LINES(3)
      WRITE(NOUT,'(''0Intermediate Dataset: '',A,/)') DSFNAME
   10 READ(LU1,'(A)',END=20) BUFFER
      CALL LINES(1)
      WRITE(NOUT,'(1X,A)') BUFFER
      GOTO 10
   20 CALL CLOSE(LU1)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
C - - Write Format(s)
      END
```

Subroutine dsgene

```
SUBROUTINE DSGENE(MDEX,NLAT,NTYPE,NRODS,RHOEFF,PELOD,MAP
     2 ,FADEX,NNUCLD,MPRFX,MSUFX,EAVG,GDCAVG,WPI,NODE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - *  D S G E N  *  Creates Fuel Material Intermediate Datasets
C - -                  for Unexposed Fuel
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Argument(s):
C - -       MDEX - Single-character Identifier for Fuel       (input)
C - -              Manufacturer
C - -               (G - GE, S - Siemens (or predecessor)
C - -                A - ABB)
```

```
C - -       NLAT - Lattice Dimensionality                        (input)
C - -      NTYPE - Number of Distinct Types of Fuel Rods         (input)
C - -      NRODS - Number of Fuel Rods in Each Rod Type          (input)
C - -     RHOEFF - Effective Density for Fuel                    (input)
C - -      PELOD - Fuel Pellet Outer Diameter (cm)               (input)
C - -        MAP - Fuel Rod Type Map                             (input)
C - -      FADEX - Index used for Lattice Identification         (input)
C - -     NNUCLD - Total Number of Entries in Valid Nuclide      (input)
C - -              List
C - -      MPRFX - Prefixes for MCNP Nuclide Identifiers in       (input)
C - -              Valid Nuclide List
C - -      MSFUX - Suffixes for MCNP Nuclide Identifiers in       (input)
C - -              Valid Nuclide List
C - -       EAVG - Lattice-averaged Enrichment (w/o)              (input)
C - -     GDCAVG - Lattice-averaged Gadolinium Concentration      (input)
C - -              (w/o)
C - -        WPI - Weight Percentages for Nuclides in Exposed (input)
C - -              Fuel (w/o)
C - -       NODE - Axial Node in Core for Lattice (or Base        (input)
C - -              for Consolidated Nodes)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  DSFNAME - File Name for Fuel Material Intermediate Dataset
C - -   BUFFER - Scratch Character String used to Manage Output to
C - -            Dataset
      INTEGER FADEX
      CHARACTER*1 MDEX
      CHARACTER*4 MSUFX(NNUCLD)
      CHARACTER*80 DSFNAME,BUFFER
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
      COMMON /FILES/ LU1,LU2,LU3
      COMMON /CODEID/ CODENM(6),TITLE,VERSON,MOD,PID
      CHARACTER*1 CODENM,MOD
      CHARACTER*2 VERSON
      CHARACTER*5 PID
      CHARACTER*122 TITLE
      COMMON /RUNINFO/ CTIME,CDATE
      CHARACTER*8 CTIME
      CHARACTER*9 CDATE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MPRFX(NNUCLD)
      DIMENSION NRODS(NTYPE),PELOD(NTYPE),WPI(NNUCLD)
      DIMENSION MAP(NLAT,NLAT)
C - - Construct File Name for Dataset - - - - - - - - - - - - - - - - -
      DSFNAME = ' '
      WRITE(DSFNAME,'(A1,I<(ALOG10(FLOAT(NLAT))+1)>)') MDEX,NLAT
      NZ = 3-(IFIX(ALOG10(100*EAVG))+1)
      IF(NZ .GT. 0) THEN
        DO N = NZ,1,-1
          WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''0'')')
        ENDDO
      ENDIF
```

```
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2 ,'(I<(ALOG10(100*EAVG)+1)>)')
     3 NINT(100*EAVG)
C - - Ensure no roundoff of character width of Gad percentages when near 1%
      IF(GDCAVG .NE. 0) THEN
        IF (GDCAVG .LT. 1.0 .AND. GDCAVG .GT. 0.995) GDCAVG=1.0
        NZ = 3-(IFIX(ALOG10(100*GDCAVG))+1)
      ELSE
        NZ = 2
      ENDIF
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''G'')')
      IF(NZ .GT. 0) THEN
        DO N = NZ,1,-1
          WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''0'')')
        ENDDO
      ENDIF
      IF(GDCAVG .NE. 0) THEN
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2    ,'(I<(ALOG10(100*GDCAVG)+1)>)')
     3    NINT(100*GDCAVG)
      ELSE
        WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2    ,'(''0'')')
      ENDIF
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''S'')')
      IF(FADEX .LT. 100)
     2  WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     3    ,'(''0'')')
      IF(FADEX .LT. 10)
     2  WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     3    ,'(''0'')')
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2 ,'(I<IFIX(ALOG10(FLOAT(FADEX)))+1>)') FADEX
      IF(NODE .LT. 10)
     2  WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     3    ,'(''0'')')
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):)
     2 ,'(I<IFIX(ALOG10(FLOAT(NODE)))+1>)') NODE
      WRITE(DSFNAME((MCHAR(80,DSFNAME)+1):),'(''.dat'')')
C - - Open File for Processing
      CALL FOPEN(LU1,DSFNAME)
C - - Write Title to Dataset
      IF(MDEX .EQ. 'G') THEN
        BUFFER = 'GE'
      ELSEIF(MDEX .EQ. 'S') THEN
        BUFFER = 'Siemens'
      ELSE
        BUFFER = 'ABB'
      ENDIF
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2  ,'('' '',I<(ALOG10(FLOAT(NLAT))+1)>,''x''
     3  ,I<(ALOG10(FLOAT(NLAT))+1)>)') NLAT,NLAT
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'(''/Avg Enrichment '',F5.2)') EAVG
```

```
      IF(GADC .NE. 0.0)
     2  WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     3   ,'(''/Avg Gadolinia '',F6.3)') GDCAVG
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2   ,'(''/Exposed Fuel'')')
      WRITE(LU1,'(A)') BUFFER
C - - Write Tracking Information to Dataset
      BUFFER = ''
      WRITE(BUFFER,'(''Generated by '',6A1)')
     2 (CODENM(I),I=1,6)
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' '',A2,'':'',A1)') VERSON,MOD
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' on '',A9,'' at '',A8)') CDATE,CTIME
      WRITE(BUFFER((MCHAR(80,BUFFER)+1):)
     2 ,'('' by Process '',A5)') PID
      WRITE(LU1,'(A)') BUFFER
C - - Fuel Rod Type Map
      WRITE(LU1,'(''Fuel Rod Type Map'')')
      DO J = 1,NLAT
        DO I = 1,NLAT
          IF(MAP(I,J) .GE. 1) MAP(I,J) = 1
        ENDDO
      ENDDO
      DO J = 1,NLAT
        WRITE(LU1,'(11(I2,'' ''))')
     2    (MAP(I,J),I=1,NLAT)
      ENDDO
C - - Density Value(s)
      WRITE(LU1,'(''Density Value(s)'')')
      WRITE(LU1,'(F6.3)') RHOEFF
C - - Material Compositions
      WRITE(LU1,'(''Fuel Material Compositions'')')
      NIT = 0
      DO N = 1,NNUCLD
        IF(WPI(N) .GT. 0.0) NIT = NIT+1
      ENDDO
      WRITE(LU1,'(I3)') NIT
      DO 10 N = 1,NNUCLD
        IF(WPI(N) .EQ. 0.0) GOTO 10
        WRITE(LU1,'(I3,'',',A4,'',',I3,'',',1P,E10.4)')
     2   (MPRFX(N)/1000),MSUFX(N)
     3   ,(MPRFX(N)-1000*(MPRFX(N)/1000)),WPI(N)
   10 CONTINUE
C - - Close File
      CALL FCLOSE(LU1)
C - - Copy File to Output - - - - - - - - - - - - - - - - - - - - -
      CALL FOPEN(LU1,DSFNAME)
      CALL HEADER
      CALL LINES(3)
      WRITE(NOUT,'(''0Intermediate Dataset: '',A,/)') DSFNAME
   20 READ(LU1,'(A)',END=30) BUFFER
      CALL LINES(1)
      WRITE(NOUT,'(1X,A)') BUFFER
```

```
      GOTO 20
   30 CALL CLOSE(LU1)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
C - - Write Format(s)
      END
```

Subroutine masses

```
      SUBROUTINE MASSES(NNUCLD,UO2NM,EAVG,RHOAVG,INM,NMEM,MPRFX
     2 ,MSUFX,WPI,FNVOL,RHOEFF)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * M A S S E S *  Edits the Nodal Nuclide Masses, Nodal UO2
C - -                    Mass, Lattice-averaged Enrichment and
C - -                    Density, and Computes Effective Density from
C - -                    SAS2H Nodal Masses and Initial Charge Oxygen
C - -                    Mass
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -       NNUCLD - Number of Nuclides Considered in Problem   (input)
C - -        UO2NM - Nodal Mass of UO2 in Problem               (input)
C - -         EAVG - Lattice-averaged Enrichment (w/o)          (input)
C - -       RHOAVG - Lattice-averaged Density (g/cc)            (input)
C - -          INM - Nodal Nuclide Mass (g)                     (input)
C - -         NMEM - SAS2H Isotope Mnemonic                     (input)
C - -        MPRFX - Prefixes for MCNP Nuclide Identifiers      (input)
C - -        MSUFX - Suffixes for MCNP Nuclide Identifiers      (input)
C - -        FNVOL - UO2 Nodal Volume (cm**3)                   (input)
C - -       RHOEFF - Effective Density (g/cc)                  (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      REAL INM(NNUCLD)
      CHARACTER*4 MSUFX(NNUCLD)
      CHARACTER*6 NMEM(NNUCLD)
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NPAGE,NIN,NOUT
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MPRFX(NNUCLD),WPI(NNUCLD)
C - - Edit Nodal UO2 Mass and Lattice-averaged Values - - - - - - - - -
      TMASS = 0
      DO N = 1,NNUCLD
        TMASS = TMASS+INM(N)
      ENDDO
      RHOEFF = TMASS/FNVOL
      CALL HEADER
      CALL LINES(9)
      WRITE(NOUT
     2 ,'(''0Total UO2 Mass Computed from SAS2H (g) = '',F10.1)')
     3 TMASS
      WRITE(NOUT
     2 ,'(''0Effective UO2 Density (g/cc) = '',F7.4)') RHOEFF
      WRITE(NOUT,
```

```
     2 '(''0Total UO2 Mass Computed from Input Lattice Values (g) = ''
     3 ,F10.1)') UO2NM
      WRITE(NOUT,'(''0Lattice-averaged Enrichment (w/o) = '',F5.3)')
     2 EAVG
      WRITE(NOUT,'(''0Lattice-averaged Density (g/cc) = '',F6.3)')
     2 RHOAVG
C - - Compute Nodal Weight Percentages -  - - - - - - - - - - - - - - -
      DO N = 1,NNUCLD
        WPI(N) = 100.0*INM(N)/UO2NM
      ENDDO
C - - Edit Nodal Masses for Nuclides - - - - - - - - - - - - - - - - - -
      CALL LINES(3)
      WRITE(NOUT,6000)
      DO N = 1,NNUCLD
        CALL LINES(1)
        WRITE(NOUT,'(1X,A6,1X,I6,A4,T20,1P,E10.4,T35,E10.4)')
     2   NMEM(N),MPRFX(N),MSUFX(N),INM(N),WPI(N)
      ENDDO
C - - End of Processing - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
C - - Write Format(s)
 6000 FORMAT('0  Identifiers     Nodal Mass  Weight Percentage',/
     2      ,'  SAS2H     MCNP        (g)            (w/o)',/)
      END
```

Subroutine match

```
      FUNCTION MATCH(QUERY,NNUCLD,MPRFX)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * M A T C H *  Matches MCNP Nuclide Prefix Query to Entry in
C - -                  Valid Nuclide List and Returns Corresponding
C - -                  Index
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        QUERY - Query MCNP Nuclide Identification Prefix   (input)
C - -       NNUCLD - Number of Nuclides Considered in Problem   (input)
C - -        MPRFX - Vector of MCNP Nuclide Identification Pre- (input)
C - -                fixes from Valid Nuclide List
C - -        MATCH - Index in Valid Nuclide List Corrsponding  (output)
C - -                to Query MCNP Nuclide Identification
C - -                Prefix
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      INTEGER QUERY
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NIN,NOUT
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MPRFX(NNUCLD)
C - - Sweep through Reference Nuclide List - - - - - - - - - - - - - - -
      MATCH = 0
      DO N = 1,NNUCLD
        IF(MPRFX(N) .EQ. QUERY) THEN
          MATCH = N
          RETURN
        ENDIF
      ENDDO
C - - Error processing - - - - - - - - - - - - - - - - - - - - - - - - -
      CALL LINES(3)
      WRITE(NOUT,7000) QUERY
      CALL ABORT
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - - Format Statement(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  Subr. MATCH --'
     2 ,' MCNP Nuclide Identifier Prefix nout Found in List'
     3 ,/,' Prefix Sought = ',A)
      END
```

Subroutine match2

```
      FUNCTION MATCH2(NNUCLD,NMEM,NAME)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  M A T C H 2 *  Matches Nuclide Labels from SAS2H Output and
C - -                     Valid Nuclide List
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -       NNUCLD - Number of Nuclides Considered in Problem   (input)
C - -         NMEM - Labels for Reference Nuclide List          (input)
C - -         NAME - Nuclide Name from SAS2H Input Deck         (input)
C - -       MATCH2 - Index in Reference List Corrsponding to   (output)
C - -                Nuclide Read from MNCP Input
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      CHARACTER*6 NMEM(NNUCLD)
      CHARACTER*6 NAME
C - - Sweep through Reference Nuclide List - - - - - - - - - - - - - - -
      MATCH2 = 0
      DO N = 1,NNUCLD
        IF(NMEM(N) .EQ. NAME) THEN
          MATCH2 = N
          RETURN
        ENDIF
      ENDDO
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine mogen

```
      SUBROUTINE MOGEN(NNUCLD,INM,UO2NM,WP25,NMEM,GDCAVG)
□
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
□
C - -   *  M O G E N *  Computes Initial Oxygen Mass for SAS2H Data
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -       NNUCLD - Number of Nuclides Considered in Problem   (input)
C - -          INM - Isotopic Nodal Mass from SAS2H Output        (i&o)
C - -        UO2NM - UO2 Nodal Mass for Fresh Fuel               (input)
C - -         WP25 - Enrichment for U-235 (Initial)              (input)
C - -         NMEM - List of SAS2H Identifers from Valid Nu-     (input)
C - -                clide List
C - -       GDCAVG - Average Gadolinia Concentration for the     (input)
C - -                Node
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -     MMGD - Atomic Weight for Gadolinia
C - -  MMGD2O3 - Molecular Weight of Gd2O3
      CHARACTER*6 NMEM(NNUCLD)
      REAL MMGD, MMGD2O3
      REAL INM(NNUCLD)
```

```
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /AWEIGHT/ AWU(4),AWGD(7),AGD(7),AWO
C - - Static Functions - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   WP24 - Computes Weight Percentage for U-234 from U-235 Weight
C - -          Percentage
C - -   WP26 - Computes Weight Percentage for U-236 from U-235 Weight
C - -          Percentage
      WP24(X) = 0.007731*(X)**1.0837
      WP26(X) = 0.0046*X
C - - Compute Weight Percentages for Uranium Isotopes - - - - - - - - - -
      E24 = WP24(WP25)
      E26 = WP26(WP25)
      E28 = 100-WP25-E24-E26
      LOC = MATCH2(NNUCLD,NMEM,' o 16 ')
      IF(LOC .EQ. 0) LOC = MATCH2(NNUCLD,NMEM,' O 16 ')
C - - Add Oxygen Inventory for UO2
      INM(LOC) =
     2 (UO2NM*2*AWO*100)
     3 /(WP25*AWU(2)+E24*AWU(1)+E26*AWU(3)+E28*AWU(4)+200*AWO)
C - - Add Oxygen Inventory for Gd2O3
      IF(GDCAVG .NE. 0) THEN
        MMGD = 0.0
        DO I = 1,7
          MMGD = AGD(I)*AWGD(I)
        ENDDO
        MMGD = MMGD/100.0
        MMGD2O3 = (2*MMGD)+(3*AWO)
        INM(LOC) = INM(LOC)+
     2    (3.0*AWO)*((GDCAVG*UO2NM/100.0)/MMGD2O3)
      ENDIF
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine sload

```
 SUBROUTINE SLOAD(LU2,NNUCLD,SFRAC,NMEM,NOGAD,NOFPGD)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  S L O A D  *  Loads Weight Fractions for Fuel Nuclides from
C - -                    SAS2H Output
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -          LU2 - Logical Unit Number for File Containing    (input)
C - -                SAS2H cut File
C - -       NNUCLD - Number of Nuclides Considered in Problem   (input)
C - -        SFRAC - Nuclide Fractions from SAS Output         (output)
C - -         NMEM - Labels for Valid Fuel Nuclides in MCNP     (input)
C - -                Deck
C - -        NOGAD - Flag to Indicate whether Gadolinia Intro-  (input)
C - -                duced as an Integral Burnable Absorber
C - -                should the Deleted from the Fuel Material
C - -                Intermediate Dataset
C - -                (0 - No, 1 - Yes)
C - -       NOFPGD - Flag to Indicate whether Fission Product   (input)
C - -                Gadolinium Isotopes are Deleted from the
```

```
C - -                 Fuel Intermediate Dataset
C - -                 (0 - No, 1 - Yes)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      LOGICAL GDFND
      REAL MASS
      CHARACTER*2 GDSYM1,GDSYM2
      CHARACTER*6 NMEM(NNUCLD)
      CHARACTER*6 NAME
      CHARACTER*132 BUFFER
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NIN,NOUT
C - - Dimension statement(s) - - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION SFRAC(NNUCLD)
C - - Data Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      DATA GDSYM1 /'gd'/, GDSYM2 /'GD'/
C - - Search for Light Nuclides in SAS2H Output - - - - - - - - - - - -
   10 READ(LU2,'(A)',END=500) BUFFER
      IF(MCHAR(132,BUFFER) .LT. 111) THEN
        GOTO 10
      ELSE
C - - Check for "light elements" Header
        IF((BUFFER(98:111) .EQ. 'light elements') .OR.
     2     (BUFFER(98:111) .EQ. 'LIGHT ELEMENTS')) THEN
C - - Read Subsequent Lines to Search for Matching Nuclides
   15     READ(LU2,'(A)') BUFFER
C - - Check for Subsequent Header Indicating End of Isotopic Mass
C - - Section
          IF((BUFFER(98:111) .EQ. 'light elements') .OR.
     2       (BUFFER(98:111) .EQ. 'LIGHT ELEMENTS')) GOTO 20
          READ(BUFFER(6:10),'(A5)') NAME
C - - Look for Gadolinia Isotopes
          IF(NOGAD .EQ. 0) THEN
            GDFND = .FALSE.
            DO I = 1,4
              IF((NAME(I:(I+1)) .EQ. GDSYM1) .OR.
     2           (NAME(I:(I+1)) .EQ. GDSYM2)) GDFND = .TRUE.
            ENDDO
            IF(GDFND) THEN
              INDEX = MATCH2(NNUCLD,NMEM,NAME)
              IF(INDEX .NE. 0) THEN
                READ(BUFFER(74:81),'(E8.2)') MASS
                SFRAC(INDEX) = SFRAC(INDEX)+MASS
              ENDIF
            ENDIF
          ENDIF
          GOTO 15
        ELSE
          GOTO 10
        ENDIF
      ENDIF
   20 ICOUNT = 0
   21 READ(LU2,'(A)',END=500) BUFFER
```

```
      IF(MCHAR(132,BUFFER) .LT. 111) THEN
        GOTO 21
      ELSE
C - - Check for "actinides" Header
        IF((BUFFER(103:111) .EQ. 'actinides') .OR.
     2     (BUFFER(103:111) .EQ. 'ACTINIDES')) THEN
          ICOUNT = ICOUNT+1
          IF(ICOUNT .LT. 3) GOTO 21
C - - Read Subsequent Lines to Search for Matching Nuclides
   25     READ(LU2,'(A)') BUFFER
          IF((BUFFER(103:111) .EQ. 'actinides') .OR.
     2       (BUFFER(103:111) .EQ. 'ACTINIDES')) GOTO 30
          READ(BUFFER(6:11),'(A6)') NAME
          INDEX = MATCH2(NNUCLD,NMEM,NAME)
          IF(INDEX .NE. 0) THEN
            READ(BUFFER(74:81),'(E8.2)') MASS
            SFRAC(INDEX) = SFRAC(INDEX)+MASS
            IF(INDEX .EQ. 19) GOTO 30
          ENDIF
          GOTO 25
        ELSE
          GOTO 21
        ENDIF
      ENDIF
   30 READ(LU2,'(A)',END=500) BUFFER
      IF(MCHAR(132,BUFFER) .LT. 111) THEN
        GOTO 30
      ELSE
C - - Check for "fission products" Header
        IF((BUFFER(96:111) .EQ. 'fission products') .OR.
     2     (BUFFER(96:111) .EQ. 'FISSION PRODUCTS')) THEN
C - - Read Subsequent Lines to Search for Matching Nuclides
   35     READ(LU2,'(A)') BUFFER
          IF((BUFFER(71:76) .EQ. 'curies') .OR.
     2       (BUFFER(71:76) .EQ. 'CURIES')) GOTO 40
          READ(BUFFER(6:11),'(A6)') NAME
          INDEX = MATCH2(NNUCLD,NMEM,NAME)
C - - Omit Gadolinium if NOFPGD Set
          IF(INDEX .NE. 0) THEN
            IF(NOFPGD .EQ. 1) THEN
              GDFND = .FALSE.
              DO I = 1,4
                IF((NAME(I:(I+1)) .EQ. GDSYM1) .OR.
     2             (NAME(I:(I+1)) .EQ. GDSYM2)) GDFND = .TRUE.
              ENDDO
              IF(GDFND) GOTO 35
            ENDIF
            READ(BUFFER(74:81),'(E8.2)') MASS
            SFRAC(INDEX) = SFRAC(INDEX)+MASS
          ENDIF
          GOTO 35
        ELSE
          GOTO 30
        ENDIF
```

```
      ENDIF
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - - -
   40 RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - - -
  500 WRITE(NOUT,7000)
      STOP
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - - Error Format(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. SLOAD ',/
     2 ,' -- Premature End-of-file Encountered')
      END
```

Subroutine sload2

```
      SUBROUTINE SLOAD2(LU2,NNUCLD,SFRAC,NMEM,NOGAD,NOFPGD)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  S L O A D  *  Loads Weight Fractions for Fuel Nuclides from
C - -                    SAS2H Output from Full Output File
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -          LU2 - Logical Unit Number for File Containing    (input)
C - -                SAS2H cut File
C - -       NNUCLD - Number of Nuclides Considered in Problem   (input)
C - -        SFRAC - Nuclide Fractions from SAS Output         (output)
C - -         NMEM - Labels for Valid Fuel Nuclides in MCNP     (input)
C - -                Deck
C - -        NOGAD - Flag to Indicate whether Gadolinia Intro-  (input)
C - -                duced as an Integral Burnable Absorber
C - -                should the Deleted from the  Fuel Material
C - -                Intermediate Dataset
C - -                (0 - No, 1 - Yes)
C - -       NOFPGD - Flag to Indicate whether Fission Product   (input)
C - -                Gadolinium Isotopes are Deleted from the
C - -                Fuel Intermediate Dataset
C - -                (0 - No, 1 - Yes)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      LOGICAL GDFND
      REAL MASS
      CHARACTER*2 GDSYM1, GDSYM2
      CHARACTER*6 NMEM(NNUCLD)
      CHARACTER*6 NAME
      CHARACTER*132 BUFFER
C - - Common Block(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
      COMMON /PAGES/ NIN,NOUT
C - - Dimension statement(s) - - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION SFRAC(NNUCLD)
C - - Data Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      DATA GDSYM1 /'gd'/, GDSYM2 /'GD'/, ICOUNT /0/
C - - Search for Light Nuclides in SAS2H Output - - - - - - - - - - - -
   10 READ(LU2,'(A)',END=500) BUFFER
      IF(MCHAR(132,BUFFER) .LT. 74) THEN
        GOTO 10
```

```
      ELSE
C - - Check for "grams" Header
        IF((BUFFER(70:74) .EQ. 'grams') .OR.
     2     (BUFFER(70:74) .EQ. 'GRAMS')) THEN
C - - Read Subsequent Lines to Search for Matching Nuclides
          ICOUNT = ICOUNT+1
          IF(ICOUNT .LT. 3) GOTO 10
   15     READ(LU2,'(A)') BUFFER
C - - Check for Subsequent Header Indicating End of Isotopic Mass
C - - Section
          IF((BUFFER(71:76) .EQ. 'curies') .OR.
     2       (BUFFER(71:76) .EQ. 'CURIES')) GOTO 20
          READ(BUFFER(6:10),'(A5)') NAME
C - - Look for Gadolinia Isotopes
          IF(NOGAD .EQ. 0) THEN
            GDFND = .FALSE.
            DO I = 1,4
              IF((NAME(I:(I+1)) .EQ. GDSYM1) .OR.
     2           (NAME(I:(I+1)) .EQ. GDSYM2)) GDFND = .TRUE.
            ENDDO
            IF(GDFND) THEN
              INDEX = MATCH2(NNUCLD,NMEM,NAME)
              IF(INDEX .NE. 0) THEN
                READ(BUFFER(74:81),'(E8.2)') MASS
                SFRAC(INDEX) = SFRAC(INDEX)+MASS
              ENDIF
            ENDIF
          ENDIF
          GOTO 15
        ELSE
          GOTO 10
        ENDIF
      ENDIF
   20 ICOUNT = 0
   21 READ(LU2,'(A)',END=500) BUFFER
      IF(MCHAR(132,BUFFER) .LT. 74) THEN
        GOTO 21
      ELSE
C - - Check for "actinides" Header
        IF((BUFFER(70:74) .EQ. 'grams') .OR.
     2     (BUFFER(70:74) .EQ. 'GRAMS')) THEN
C - - Read Subsequent Lines to Search for Matching Nuclides
   25     READ(LU2,'(A)') BUFFER
          IF((BUFFER(46:52) .EQ. 'element') .OR.
     2       (BUFFER(46:52) .EQ. 'ELEMENT')) GOTO 30
          READ(BUFFER(6:11),'(A6)') NAME
          INDEX = MATCH2(NNUCLD,NMEM,NAME)
          IF(INDEX .NE. 0) THEN
            READ(BUFFER(74:81),'(E8.2)') MASS
            SFRAC(INDEX) = SFRAC(INDEX)+MASS
          ENDIF
          GOTO 25
        ELSE
          GOTO 21
```

```
        ENDIF
      ENDIF
   30 READ(LU2,'(A)',END=500) BUFFER
      IF(MCHAR(132,BUFFER) .LT. 74) THEN
        GOTO 30
      ELSE
C - - Check for "fission products" Header
        IF((BUFFER(70:74) .EQ. 'grams') .OR.
     2     (BUFFER(70:74) .EQ. 'GRAMS')) THEN
C - - Read Subsequent Lines to Search for Matching Nuclides
   35     READ(LU2,'(A)') BUFFER
          IF((BUFFER(71:76) .EQ. 'curies') .OR.
     2       (BUFFER(71:76) .EQ. 'CURIES')) GOTO 40
          READ(BUFFER(6:11),'(A6)') NAME
          INDEX = MATCH2(NNUCLD,NMEM,NAME)
          IF(INDEX .NE. 0) THEN
            IF(NOFPGD .EQ. 1) THEN
              GDFND = .FALSE.
              DO I = 1,4
                IF((NAME(I:(I+1)) .EQ. GDSYM1) .OR.
     2              (NAME(I:(I+1)) .EQ. GDSYM2)) GDFND = .TRUE.
              ENDDO
              IF(GDFND) GOTO 35
            ENDIF
            READ(BUFFER(74:81),'(E8.2)') MASS
            SFRAC(INDEX) = SFRAC(INDEX)+MASS
          ENDIF
          GOTO 35
        ELSE
          GOTO 30
        ENDIF
      ENDIF
C - - End of processing - - - - - - - - - - - - - - - - - - - - - - - - - -
   40 RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - - - -
  500 WRITE(NOUT,7000)
      STOP
C - - FORMAT Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - - Error Format(s)
 7000 FORMAT('0***  F A T A L  E R R O R  ***  SUBR. SLOAD2 ',/
     2 ,' -- Premature End-of-file Encountered')
      END
```

CONTENTS

	Page
1. Introduction	4
2. Specifications	5
2.1. Definition of Fuel Rods	5
2.2. Definition of Water Rod	5
2.3. Definition of Fuel Lattice	5
2.4. Definition of Fuel Channel	5
2.5. Integration of Components	5
3. Encoding of Process	12
3.1. Driver Function	12
3.2. Lattice-preparation Function	12

FIGURES

		Page
2-1	Fuel Rod	6
2-2	Lattice Population	7
2-3	Integration of Components into Lattice	8
3-1	Flowchart for Creation of GE 8x78 Lattice with Large Central Water Rod	13

WORKSHEETS

	Page
2-1 Computation of Surface Coordinates	9
2-2 Cell Definitions	11

1. Introduction

This attachment describes the methodology used to create GE 8x8 (large central water rod) fuel lattice models for use in MCNP (References 7.1, 7.2, 7.5, and 7.6) representations of Boiling Water Reactor (BWR) cores. The methodology assumes the existence of a dataset describing the blade geometry that has been generated according to the requirements of Attachment V.

The driver coding within the linkage automation is intended to be sufficiently robust to readily model all the varieties BWR fuel and is described in Attachment X. This attachment describes the modeling of a GE 8x8 fuel lattice with a large central water rod. Such a water rod displaces the four central fuel rods in the lattice.

2. Specifications

This process is a subset of the larger process described and illustrated in Attachment X and this attachment describes the particular processing for a GE 8x8 fuel lattice with a large central rod. The basis values used in this attachment are shown in Attachment IV. Note that the values shown herein do not necessarily correspond to a particular lattice used in the analyses, but are shown to illustrate the process whereby such an MCNP is constructed.

2.1. Definition of Fuel Rods.

The fuel rods are comprised of zircaloy tubes filled with UO_2 ceramic pellets (Reference 7.8) as shown in Figure 2-1. Computation of the values for the absorber tubes defining surfaces is shown in Worksheet 2-1, while the cell and universe definitions are shown in Worksheet 2-2.

2.2. Definition of Water Rod

The large central water rod is a hollow zircaloy tube through which un-voided water flows (Reference 7.4). Computation of the values for the absorber tubes defining surfaces is shown in Worksheet 2-1, while the cell and universe definitions are shown in Worksheet 2-2.

2.3. Definition of Fuel Lattice

The fuel rods fill a regular lattice. In the case of the GE 8x8 fuel design with small water rods, one or more of these fuel rods are displaced with cladding hulls through which non-voided water flows (see Figure 2-2). Note that in the model four representations of the water rod are produced with different center coordinates. This permits the water rod to be assembled from these four copies as filling universes in the lattice defining the fuel lattice.

2.4. Definition of Fuel Channel

The channel is assumed to be of uniform thickness with rounded corners and is as described in Attachment X for a "D" lattice geometry.

2.5. Integration of Components

The combination of the lattice components into a complete model is illustrated in Figure 2-3.

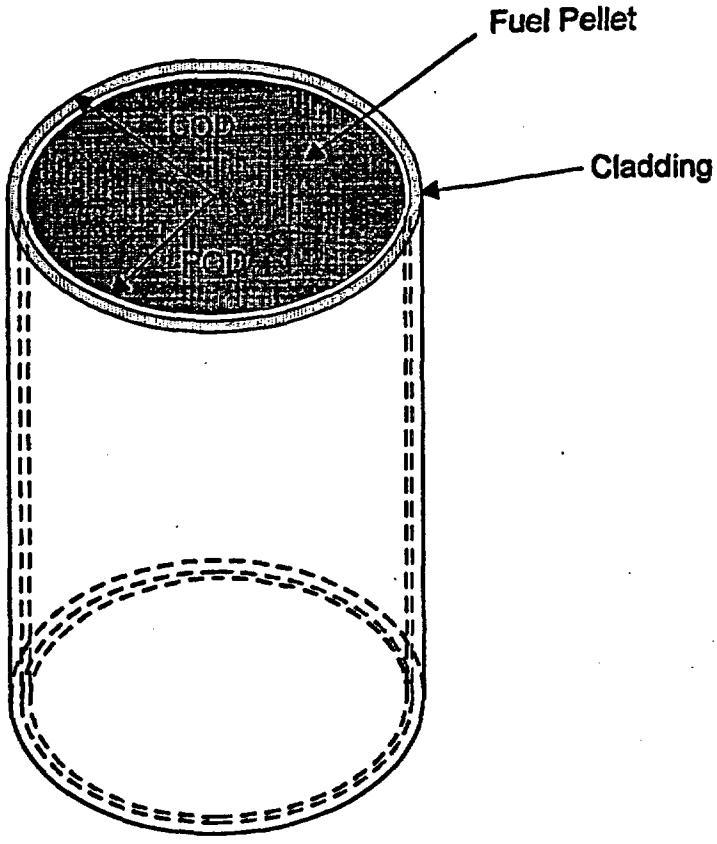

Figure 2-1 Fuel Rod

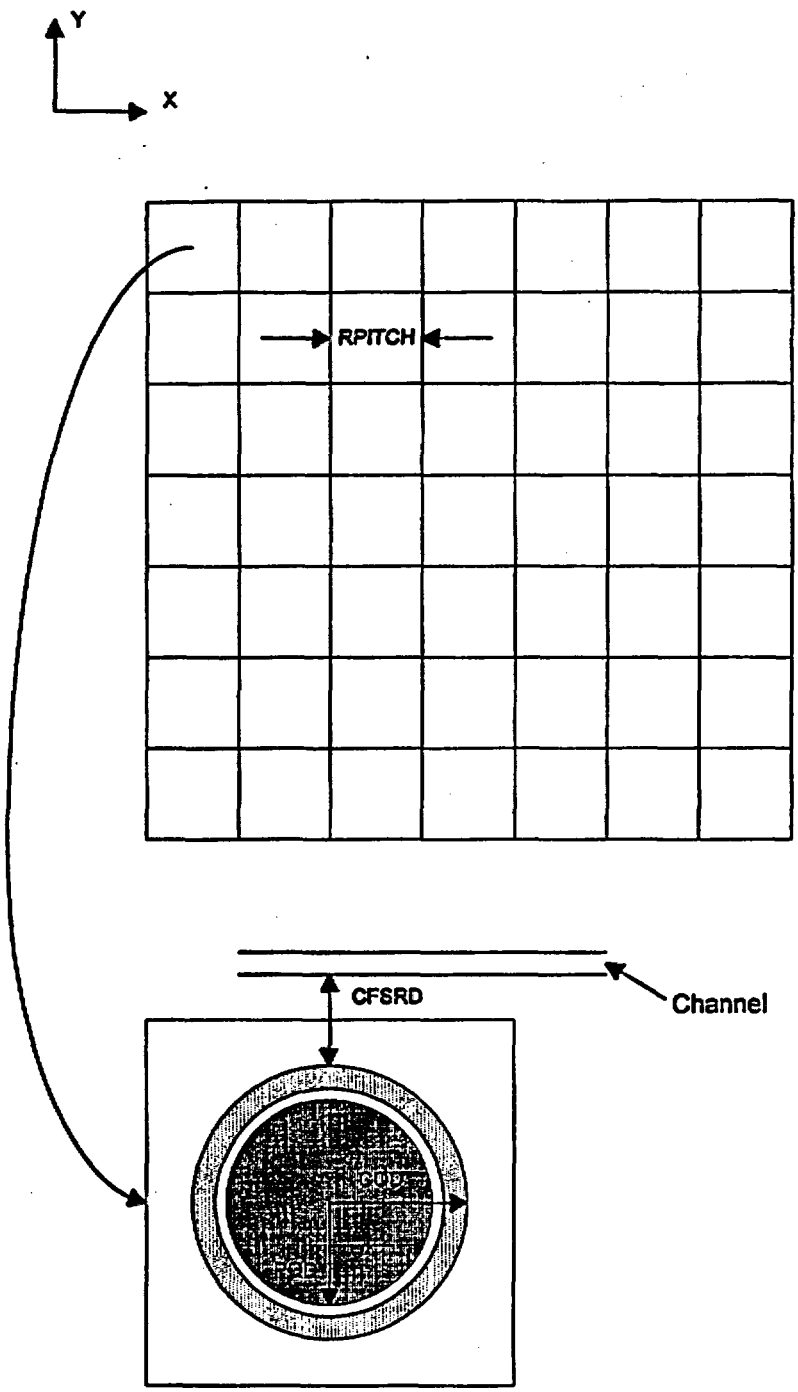

Figure 2-2 Lattice Population

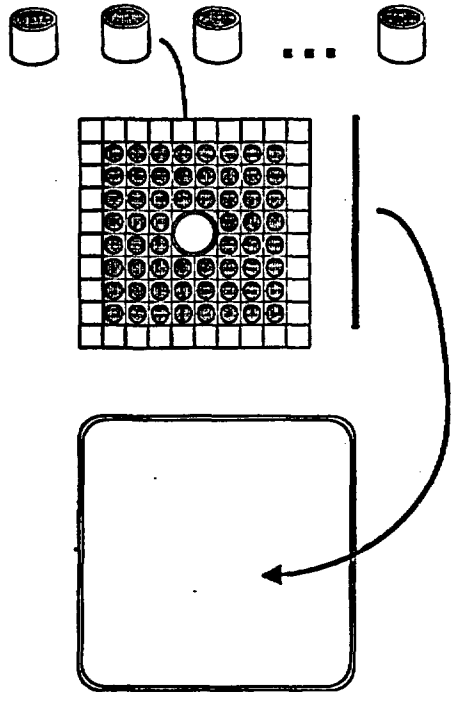

Figure 2-3 Integration of Components into Lattice

Worksheet 2-1 Computation of Surface Coordinates

Index	Symbol	Definition	Mnemonic	Parameters	Computation
1	SPOR	Reference Fuel Pellet Outer Surface	c/z	2.15645	= WGAP+CTHICK+CFSRD+(COD/2)
				-2.15645	= -(WGAP+CTHICK+CFSRD+(COD/2))
				0.5207	= POD/2
2	SCIR	Reference Cladding Inner Surface	c/z	2.15645	= WGAP+CTHICK+CFSRD+(COD/2)
				-2.15645	= -(WGAP+CTHICK+CFSRD+(COD/2))
				0.5321	= (COD/2)-CLD
3	SCOR	Reference Cladding Outer Surface	c/z	2.15645	= WGAP+CTHICK+CFSRD+(COD/2)
				-2.15645	= -(WGAP+CTHICK+CFSRD+(COD/2))
				0.6134	= COD/2
4	SIWRNWQ	Water Rod Inner Surface (NW Quadrant)	c/z	2.96925	= WGAP+CTHICK+CFSRD+(COD/2)+(RPITCH/2)
				-2.96925	= -(WGAP+CTHICK+CFSRD+(COD/2)+(RPITCH/2))
				1.2281	= WRID/2
5	SOWRNWQ	Water Rod Outer Surface (NW Quadrant)	c/z	2.96925	= WGAP+CTHICK+CFSRD+(COD/2)+(RPITCH/2)
				-2.96925	= -(WGAP+CTHICK+CFSRD+(COD/2)+(RPITCH/2))
				1.3094	= WORD/2
6	SIWRNEQ	Water Rod Inner Surface (NE Quadrant)	c/z	1.34365	= WGAP+CTHICK+CFSRD+(COD/2)-(RPITCH/2)
				-2.96925	= -(WGAP+CTHICK+CFSRD+(COD/2)+(RPITCH/2))
				1.2281	= WRID/2
7	SOWRNEQ	Water Rod Outer Surface (NE Quadrant)	c/z	1.34365	= WGAP+CTHICK+CFSRD+(COD/2)-(RPITCH/2)
				-2.96925	= -(WGAP+CTHICK+CFSRD+(COD/2)+(RPITCH/2))
				1.3094	= WORD/2
8	SIWRSEQ	Water Rod Inner Surface (SE Quadrant)	c/z	1.34365	= WGAP+CTHICK+CFSRD+(COD/2)-(RPITCH/2)
				-1.34365	= -(WGAP+CTHICK+CFSRD+(COD/2)-(RPITCH/2))
				1.2281	= WRID/2
9	SOWRSEQ	Water Rod Outer Surface (SE Quadrant)	c/z	1.34365	= WGAP+CTHICK+CFSRD+(COD/2)-(RPITCH/2)
				-1.34365	= -(WGAP+CTHICK+CFSRD+(COD/2)-(RPITCH/2))
				1.3094	= WORD/2
10	SIWRSWQ	Water Rod Inner Surface (SW Quadrant)	c/z	2.96925	= WGAP+CTHICK+CFSRD+(COD/2)+(RPITCH/2)
				-1.34365	= -(WGAP+CTHICK+CFSRD+(COD/2)-(RPITCH/2))
				1.2281	= WRID/2
11	SOWRSWQ	Water Rod Outer Surface (SW Quadrant)	c/z	2.96925	= WGAP+CTHICK+CFSRD+(COD/2)+(RPITCH/2)
				-1.34365	= -(WGAP+CTHICK+CFSRD+(COD/2)-(RPITCH/2))

Index	Symbol	Definition	Mnemonic	Parameters	Computation
				1.3094	= WORD/2
12	XMINFRW	XMIN Surface for Fuel Rod Window	px	-0.8128	= -(RPITCH/2)
13	XMAXFRW	XMAX Surface for Fuel Rod Window	px	0.8128	= RPITCH/2
14	YMINFRW	XMIN Surface for Fuel Rod Window	py	-0.8128	= -(RPITCH/2)
15	YMAXFRW	XMAX Surface for Fuel Rod Window	py	0.8128	= RPITCH/2
16	WGCOWX	Wide Gap, Channel Outside Wall, pX Surface	px	0.9525	= WGAP
17	WGCIWX	Wide Gap, Channel Inside Wall, pX Surface	px	1.1557	= WGAP+CTHICK
18	NGCIWX	Narrow Gap, Channel Inside Wall, pX Surface	px	14.56182	= WGAP+CTHICK+ASIN
19	NGCOWX	Narrow Gap, Channel Outside Wall, pX Surface	px	14.76502	= WGAP+(2*CTHICK)+ASIN
20	WGCOWY	Wide Gap, Channel Outside Wall, pY Surface	py	-0.9525	= -WGAP
21	WGCIWY	Wide Gap, Channel Inside Wall, pY Surface	py	-1.1557	= -(WGAP+CTHICK)
22	NGCIWY	Narrow Gap, Channel Inside Wall, pY Surface	py	-14.56182	= -(WGAP+CTHICK+ASIN)
23	NGCOWY	Narrow Gap, Channel Outside Wall, pY Surface	py	-14.76502	= -(WGAP+(2*CTHICK)+ASIN)
24	XAMBIG1	Ambiguity Surface for Channel Corners (Wide Gap)	px	2.1717	= WGAP+CTHICK+CRADIUS
25	XAMBIG2	Ambiguity Surface for Channel Corners (Narrow Gap)	px	13.54582	= WGAP+CTHICK+ASIN-CRADIUS
26	YAMBIG1	Ambiguity Surface for Channel Corners (Wide Gap)	py	-2.1717	= -(WGAP+CTHICK+CRADIUS)
27	YAMBIG2	Ambiguity Surface for Channel Corners (Narrow Gap)	py	-13.54582	= -(WGAP+CTHICK+ASIN-CRADIUS)
28	CC1RO	Outer Radius for Corner 1	c/z	2.1717	= WGAP+CTHICK+CRADIUS
				-2.1717	= -(WGAP+CTHICK+CRADIUS)
				1.2192	= CRADIUS+CTHICK
29	CC1RI	Inner Radius for Corner 1	c/z	2.1717	= WGAP+CTHICK+CRADIUS
				-2.1717	= -(WGAP+CTHICK+CRADIUS)
				1.016	= CRADIUS
30	CC2RO	Outer Radius for Corner 2	c/z	13.54582	= WGAP+CTHICK+ASIN-CRADIUS
				-2.1717	= -(WGAP+CTHICK+CRADIUS)
				1.2192	= CRADIUS+CTHICK
31	CC2RI	Inner Radius for Corner 2	c/z	13.54582	= WGAP+CTHICK+ASIN-CRADIUS
				-2.1717	= -(WGAP+CTHICK+CRADIUS)
				1.016	= CRADIUS
32	CC3RO	Outer Radius for Corner 3	c/z	13.54582	= WGAP+CTHICK+ASIN-CRADIUS
				-13.54582	= -(WGAP+CTHICK+ASIN-CRADIUS)
				1.2192	= CRADIUS+CTHICK
33	CC3RI	Inner Radius for Corner 3	c/z	13.54582	= WGAP+CTHICK+ASIN-CRADIUS
				-13.54582	= -(WGAP+CTHICK+ASIN-CRADIUS)
				1.016	= CRADIUS
34	CC4RO	Outer Radius for Corner 4	c/z	2.1717	= WGAP+CTHICK+CRADIUS
				-13.54582	= -(WGAP+CTHICK+ASIN-CRADIUS)

Index	Symbol	Definition	Mnemonic	Parameters	Computation
				1.2192	= CRADIUS+CTHICK
35	CC4RI	Inner Radius for Corner 4	c/z	2.1717	= WGAP+CTHICK+CRADIUS
				-13.54582	= -(WGAP+CTHICK+ASIN-CRADIUS)
				1.016	= CRADIUS

Worksheet 2-2 Cell Definitions

Index	Symbol	Universe	Symbol	Definition	Cell Definition
1	CFP	1	UFR	Fuel Pellet	-1 u= 1
2	CFCG	1		Pellet-Cladding Gap	-2 1 u= 1
3	CFRC	1		Cladding	-3 2 u= 1
4	CMOFR	1		Moderator Outside Fuel Rod	3 u= 1
5	CIWRNW	2	UWRNW	Water Inside Water Rod (NW)	-4 u= 2
6	CWRNW	2		Water Rod (NW)	-5 4 u= 2
7	CMOWRNW	2		Moderator Outside Water Rod (NW)	5 u= 2
8	CIWRNE	3	UWRNE	Water Inside Water Rod (NE)	-6 u= 3
9	CWRNE	3		Water Rod (NE)	-7 6 u= 3
10	CMOWRNE	3		Moderator Outside Water Rod (NE)	7 u= 3
11	CIWRSE	4	UWRSE	Water Inside Water Rod (SE)	-8 u= 4
12	CWRSE	4		Water Rod (SE)	-9 8 u= 4
13	CMOWRSE	4		Moderator Outside Water Rod (SE)	9 u= 4
14	CIWRSW	5	UWRSW	Water Inside Water Rod (SW)	-10 u= 5
15	CWRSW	5		Water Rod (SW)	-11 10 u= 5
16	CMOWRSW	5		Moderator Outside Water Rod (SW)	11 u= 5
17	CWFRL	6	UFRL	Window for Fuel Rod (Lattice)	12 -13 14 -15 u= 6 fill= 1,2,3,4 or 5
18	CCHAN	7	UCHAN	Channel	(16 -17 -26 27):(-20 21 24 -25): (-22 23 24 -25):(18 -19 -26 27): (-28 29 26 -24):(-30 31 26 25): (-32 33 25 -27):(-34 35 -27 -24) u= 7
19	CWIC	7		Water Inside Channel	(24 -25 -21 22):(17 -24 -26 27): (25 -18 -26 27):(-29 -24 26): (-31 25 27):(-33 25 -27):(35 -24 -27) u= 7 fill= 6
20	CWOC	7		Water Outside Channel	#19 #18 u= 7

3. Encoding of Process

There are two distinct parts of the process for creating lattice models. The first is a driver function that manages the selection of the appropriate data for the lattice and second is a lattice-geometry-specific function that creates the detailed lattice model.

3.1. Driver Function

A description of this function is provided in Attachment X.

3.2. Lattice-preparation Function

A global description of this function is provided in Attachment X. The flowchart of this process for GE 8x8 fuel assemblies with small water rods is shown in Figure 3-1.

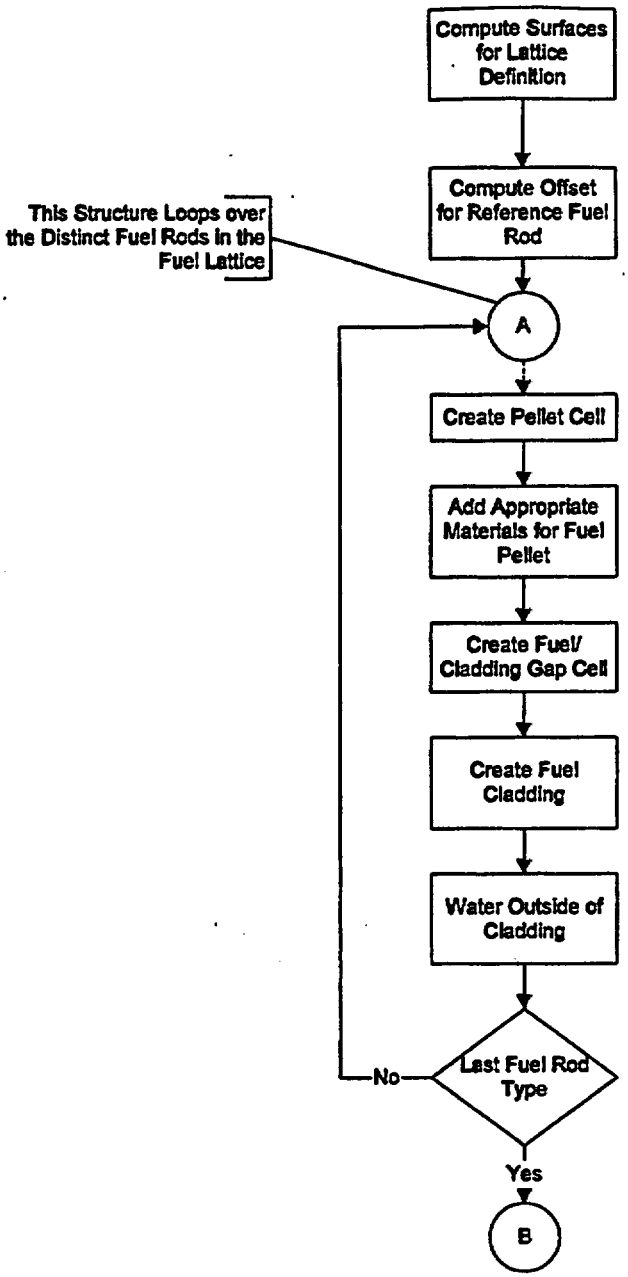

Figure 3-1 Flowchart for Creation of GE 8x8 Lattice with Large Central Water Rod

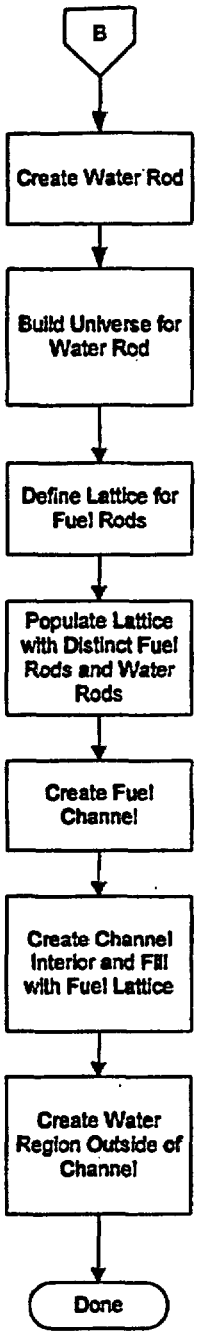

Figure 3-1 (cont'd)

CONTENTS

	Page
1. Introduction	4
2. Descriptions of Changes	4
2.1. Varying-length Axial Fuel Nodes	4
2.2. Changes to Input Instructions	6
2.3. Program Components Modified and Added	9
3. Integration Testing	13

FIGURES

	Page
2-1 New Control Cell Construction Scheme	5
2-2 BLINK Input Deck for Quad Cities Unit-1 Initial Core	8

TABLES

	Page
2-1 Input Variables	6
2-2 New or Modified Service Routines	10
2-3 New or Modified Input Routines	11
2-4 New or Modified Input Editing Routines	11
2-5 New or Modified Deck Generation Routines	12

1. Introduction

This attachment describes the creation of version 1 of the BLINK linkage automation software routine. This is a revision of the first version of this software routine which is described in Attachment VI.

2. Descriptions of Changes

The substantive changes to the software routine are:

1) added the capability to model axial nodes in the fuel assembly with differing height;
2) added a model for GE 8x8 fuel lattices incorporating a Large Central Water Rod; and
3) changed the construction of the control cell to reduce the number of surfaces to provide headroom to MCNP input limitations.

Each of these changes will now be described in greater detail.

2.1. Varying-length Axial Fuel Nodes

The ability to select differing lengths for the axial nodes in fuel assembly models was achieved by incorporating new input variables that specify the top of each axial node for each geometrically unique fuel assembly type in the core. The top and bottom of the fuel assembly are restricted to be the problem datum of the bottom and the axial location corresponding to a distance above the datum equivalent to the axial fuel length (the variable "afl").

2.1.1. Addition of GE 8x8 Lattice Design with Large Central Water Rod

The construction of the model for a GE 8x8 lattice design incorporating a large central water rod is described in Attachment XVI.

2.1.2. Changes to Control Cell Construction

The existing control cell model, as depicted schematically in Figure 3-6 of Attachment VI, required the use of many surfaces within MCNP and challenged the limitations of the input processor (Reference 7.1, page 3-11); therefore, the bounding surfaces were changed to be those shown in Figure 2-1. This change reduced the total number of surfaces and simplified the specification of the cell representing the control cell.

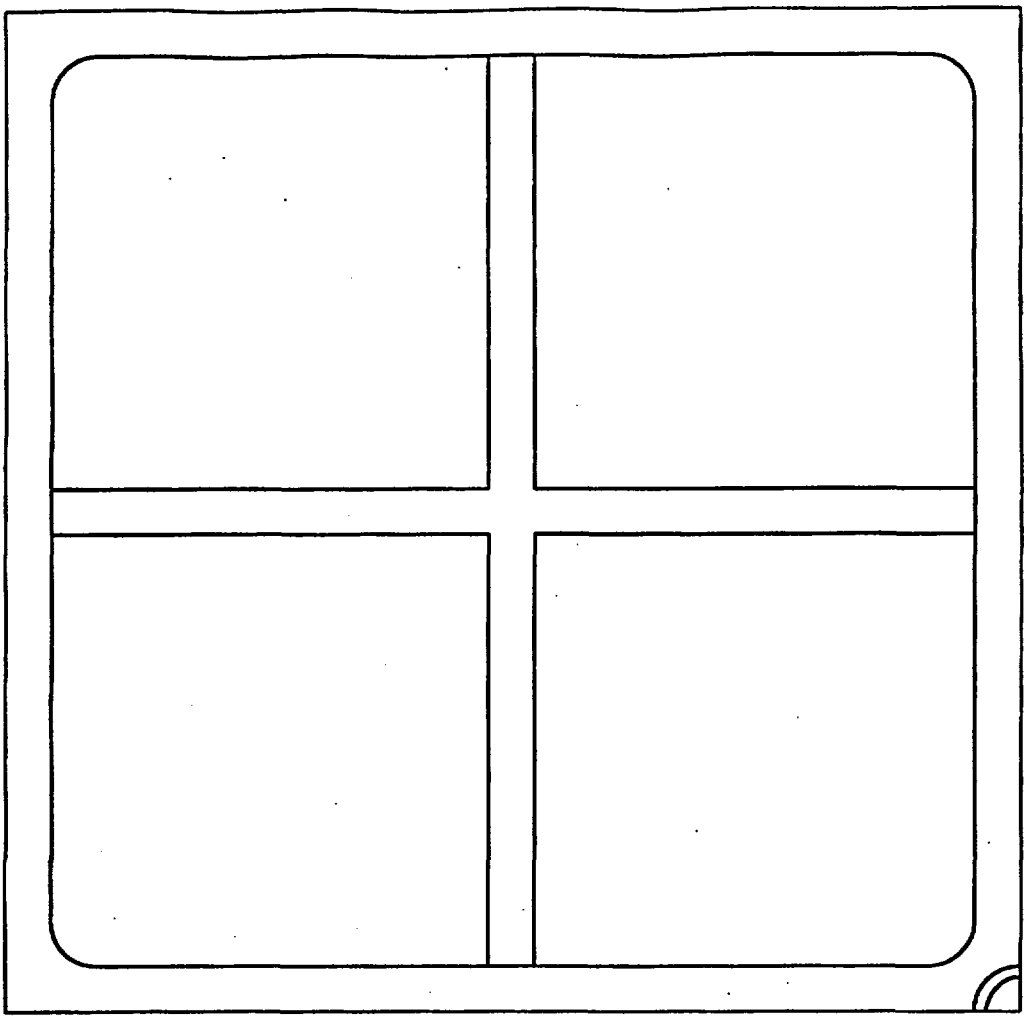

Figure 2-1 New Control Cell Construction Scheme

2.2. Changes to Input Instructions

The only changes to the software routine was the introduction of the variables controlling the node height; nevertheless, all of the input instructions will be provided here for completeness. A copy of the input deck for the Quad Cities Unit-1 initial core with the input for assembly-specific node sizes is shown in Figure 2-2.

2.2.1. Dataset Title Record

The first line of the dataset is a title. While the contents of this line are arbitrary, good practice indicates that it should contain the following information:

- name of plant modeled;
- cycle and exposure point;
- thermal-hydraulic conditions; and
- software routine execution options.

2.2.2. Namelist Input

The FORTRAN Namelist-type input variables must adhere to the restrictions inherent in the format of such input. Care must be taken to ensure that the value provided is consistent with the data storage class in the automation (i.e., integer input for integer variables and real input for real variables) so that neither precision is not lost for real variables nor is illusory precision implied for integer variables.

2.2.3. List Input Fields

These fields are used for vectors and arrays of data, such as indices to fuel assembly axial nodes and "maps" indicating the locations of fuel assembly geometrical types. While these are read in a "free-format," good practice indicates that they should be arrayed in a regular fashion that maximizes legibility. The list input fields are preceded by a title line in every instance.

Table 2-1 Input Variables

Variable	Definition	Comments	Format
Title	Case Title	Character String -- Describes Analysis	Single Line
CORE_DB	File Specification for Core Geometry Database		Namelist
CORE_MTLS	File Specification for Core Materials Database		Namelist
BLADE_DB	File Specification for Control Blade Geometry Database		Namelist
FPREFIX	Directory Specification for Location of Fuel Material Intermediate Database		Namelist
LPREFIX	Directory Specification for Location of Lattice Geometry Database		Namelist
NAXIAL	Number of Axial Nodes in Core Representation	Must be Consistent with SAS2H Analysis forming the Source of the Fuel Material Compositions	Namelist

Variable	Definition	Comments	Format
AFL	Active Fuel Length	Value must be in centimeters	Namelist
NCOLP	Maximum Number of Columns of Fuel Assemblies in the Problem	May be Consistent with Quarter-core, Half-core or Full-core Representations	Namelist
NROWP	Maximum Number of Rows of Fuel Assemblies in the Problem	May be Consistent with Quarter-core, Half-core or Full-core Representations	Namelist
RHO	Density for In-channel Moderator	Units are g/cm^3	Namelist
RHOBYP	Density for Moderator in Bypass Region	Units are g/cm^3	Namelist
TEMPK	Problem Temperature for Scattering Kernel	Units are Kelvin	Namelist
MUTP	Material Identifier for Upper Tie Plate Region	Maximum of Six Characters	Namelist
MLTP	Material Identifier for Lower Tie Plate Region	Maximum of Six Characters	Namelist
GMAP	Map Pointing to Fuel Assembly Geometrical Dataset Vectors		List Format
MMAP	Map Pointing to Fuel Material Intermediate Dataset Vectors		List Format
BLADEP	Control Blade Position	Map of Control Blade Positions (must be an even integer between 0 and 48, inclusive)	List Format
LGVECT	Lattice Geometry Vectors	Supplies Description of Datasets that Represent the Geometry of the Lattices	List Format
LMVECT	Lattice Material Vectors	Supplies Description of Datasets that Represent the Material Composition of the Lattices	List Format
N_NODE	Number of Axial Nodes for Each Geometrical Fuel Type		List Format
NODE_ BOUNDARIES	Tops of Axial Nodes	Units are in cm	List Format
N_SPACER	Number of Spacers for each Geometrical Fuel Type		List Format
S_LOC	Spacer Location for Each Spacer for a Given Geometrical Fuel Type		List Format
S_MTL	Spacer Material Label for Each Geometrical Fuel Type		List Format

```
Quad Cities-1, Beginning of Life
$LINKIN
 CORE_DB = '/home/anderson/crc_bwr/core_database/bwr3_724bundle.dat'
 CORE_MTLS = '/home/anderson/crc_bwr/materials_database/core_materials.dat'
 BLADE_DB = '/home/anderson/crc_bwr/blade_database/ge_d_lattice.dat'
 LPREFIX = '/home/anderson/crc_bwr/lattice_database/'
 FPREFIX = '/home/anderson/crc_bwr/qclic/fuel_composition_database/'
 NAXIAL = 24
 AFL = 365.760
 NCOLP = 15
 NROWP = 15
 RHO = 0.981141
 RHOBYP = 0.981141
 TEMPK = 337.04
 MUTP = '7GUTP1'
 MLTP = '7GLTP1'
$END
Fuel Geometry Loading Map
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Fuel Material Loading Map
 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3
 0 0 0 0 0 0 0 0 0 3 3 1 3 1 3
 0 0 0 0 0 0 3 3 3 3 1 4 1 3 1
 0 0 0 0 0 3 3 1 3 1 4 1 4 2 4
 0 0 0 0 3 3 1 3 2 4 2 4 2 4 2
 0 0 0 3 3 1 3 2 4 2 4 1 4 2 4
 0 0 3 3 1 3 1 4 2 4 2 4 2 4 2
 0 0 3 1 3 2 4 2 4 2 4 1 3 1 3
 0 0 3 3 2 4 2 4 2 4 1 3 1 3 1
 0 3 3 1 4 2 4 2 4 1 3 1 3 1 3
 3 3 1 4 2 4 2 4 1 3 1 3 1 3 1
 3 1 3 2 4 2 4 1 3 1 3 1 3 1 3
 3 3 1 4 2 4 2 3 1 3 1 3 1 3 1
 3 1 3 2 4 2 4 1 3 1 3 1 3 1 3
 3 3 1 4 2 4 2 3 1 3 1 3 1 3 1
```

Figure 2-2 BLINK Input Deck for Quad Cities Unit-1 Initial Core

```
Blade Positions
 -1 -1 -1 -1 -1 48 00 48
 -1 -1 -1 00 00 00 00 00
 -1 -1 00 48 00 48 00 48
 -1 00 00 00 00 00 00 00
 -1 48 00 48 00 48 00 48
 00 00 00 00 00 00 00 00
 00 48 00 48 00 48 00 48
 00 00 00 00 00 00 00 00
Lattice Geometry Indices
 1 24*1
Lattice Material Indicies
 1 1 7*2 10*3 5*2 1
 2 4 7*5 10*6 5*5 4
 3 7 22*8 7
 4 9 22*10 9
Lattice Geometry Datasets
ge7x7.dat
Lattice Material Datasets
G7212G003DL1.dat
G7212G006DL2.dat
G7212G007DL3.dat
G7211G003DL4.dat
G7211G006DL5.dat
G7211G007DL6.dat
G7212G003DL7.dat
G7212G006DL8.dat
G7211G003DL9.dat
G7212G006DL10.dat
Number of Fuel Node Surfaces for each Bundle Type
23
Tops of Fuel Nodes for each Bundle Type
15.24 30.48 45.72 60.96 76.20 91.44 106.68 121.92 137.16 152.40
 167.64 182.88 198.12 213.36 228.60 243.84 259.08 274.32 289.56
 304.80 320.04 335.28 350.52
Fuel Assembly Spacers for each Bundle Type
7
Locations of Spacer for Each Bundle Type
46.99 96.52 146.05 195.58 245.11 294.64 344.17
Spacer Material Mnemonics
SG7D1
```

FIGURE 2.2 (cont'd)

2.3. Program Components Modified and Added

Many program components – both FORTRAN routines and C functions – were modified, and in some cases added, as a result of the creation of the new version. The descriptions of the new or altered components are discussed in the same structure as used in Attachment VI. Note that some components were re-compiled as a result of testing performed on the new version of the software routine and are essentially unchanged from the previous version.

2.3.1. Driver Routine

The main C function for the linkage automation software routine was modified to accommodate the changes described above.

2.3.2. Service Routines

These are routines that provide memory management, file management, control of overall output processing, and miscellaneous services. The new or modified components are shown in Table 2-2.

Table 2-2 New or Modified Service Routines

Name [a]	Function	Description of Change
load_core_mtls (c)	Loads the Contents of the Core Materials Dataset into Memory	Unchanged
load_fuel_material (c)	Loads the Contents of a Intermediate Fuel Material Dataset in Memory	Altered to Accommodate the Water Rod Dimensions
load_surface_usage_list (c)	Loads Entries into "surface_usage_list" Structure in Linked List	Unchanged
memory_ascii_record (c)	Manages Memory Requests for Variables of the "ascii_record" Type	Unchanged
memory_fg_list (c)	Manages Memory Requests for Variables of the "fg_list" Type	Altered to Accommodate the Water Rod Dimensions
memory_float (c)	Manages Memory Requests for Single-precision Real Variables	Unchanged
memory_integer (c)	Manages Memory Requests for Integer Variables	Unchanged
memory_lattice_list (c)	Manages Memory Requests for Variables of the "augmented_lattice_list"	Unchanged
memory_s_material (c)	Manages Memory Requests for Variables of the "ll_material" Type	Unchanged
memory_surface_usage_list (c)	Manages Memory Requests for Variables of the "surface_usage_list" Type	Unchanged
memory_usage_list (c)	Manages Memory Requests for Variables of the "usage_list" Type	Unchanged
search_surface_usage_list (c)	Searches Linked Lists of the "surface_usage_list" Type either by Index or Label	Unchanged

[a]. The character in parentheses represents the computer language in which the routine is coded. A lower case "c" represents C source statements while a lower case "f" represents FORTRAN source statements. The name of FORTRAN source routines are also given in all uppercase letters.

2.3.3. Input Processing

These routines control the processing of input data to the software routine. Thus, they process the input variables shown in Table 2-1. The new or modified components are shown in Table 2-3.

Table 2-3 New or Modified Input Routines

Name [a]	Function	Description of Change
LDLV (f)	Reads Vectors of Integers and an Associated Title Line	Unchanged
LODCT (f)	Creates Correspondence Table between Lattice Geometry Indices and Lattice Material Indices	Unchanged
RBLADE (f)	Reads the Contents of the Control Blade Geometry Dataset	Unchanged
READIN (f)	Manages the Reading of Input Files	Modified to Accommodate New Variables
RLATTICE (f)	Reads the Contents of the Lattice Geometry Dataset	Modified to Accommodate Water Rod Dimensions

[a]. The character in parentheses represents the computer language in which the routine is coded. A lower case "c" represents C source statements while a lower case "f" represents FORTRAN source statements. The name of FORTRAN source routines are also given in all uppercase letters.

2.3.4. Input Editing Routines

These routines edit the user input to the software routine and the contents of many of the datasets selected for the creation of the MCNP input deck. The new or modified components are shown in Table 2-4.

Table 2-4 New or Modified Input Editing Routines

Name [a]	Function	Description of Change
fgds_edt (c)	Edits Contents of Fuel Geometry Datasets	Modified to Accommodate Water Rod Dimensions
edit_ct (c)	Edits Correspondence Table which Relates Lattice Geometry Indices and Lattice Material Indices	Unchanged
edit_spacer (c)	Edits Input Variables Defining Fuel Assembly Spacers	Unchanged

[a]. The character in parentheses represents the computer language in which the routine is coded. A lower case "c" represents C source statements while a lower case "f" represents FORTRAN source statements. The name of FORTRAN source routines are also given in all uppercase letters.

2.3.5. MCNP Input Deck Generation

These components generate the input representations for MCNP which represent the core and its components. The new or modified components are shown in Table 2-5.

Table 2-5 New or Modified Deck Generation Routines

Name [a]	Function	Description of Change
add_surface (c)	Adds Surface Definition to MCNP Model	Unchanged
augment_lattice_list (c)	Adds Lattices Incorporating Spacer Grids into Material Lattice Loading Vectors	Unchanged
build_assemblies (c)	Combines Unique Lattices Type into Unique Assembly Types	Unchanged
ge7x7_lattice (c)	Creates Cells, Surfaces and Material Definitions for Model of GE 7x7 Lattice (No Water Rods)	Unchanged
ge8x8_lattice_swr (c)	Creates Cells, Surfaces and Material Definitions for Model of GE 8x8 Lattice with Small Water Rods	Corrected Water Density Outside of Water Rod
ge8x8_lattice_lcwr (c)	Creates Cells, Surfaces and Material Definitions for Model of GE 8x8 Lattice with Large Central Water Rods	New Function
generate_lattice_model (c)	Controls the Generation of the Appropriate Lattice Representation for each Unique Lattice in the Core [b]	Added Function Call for GE 8x8 Lattice with Large Central Water Rod
material_match (c)	Matches Material Identifiers with Materials in the Linked List for Core Materials	Unchanged
source_specification (c)	Adds Source Specification and Other Problem Control to the MCNP Input	Changed Specification for Edit Vector
spacer_location (c)	Determines the Nodal Locations of Each Fuel Spacer Grid	Adjustments to Accommodate Variable-length Axial Fuel Nodes
vessel_generation (c)	Creates Cells, Surfaces and Material Definitions for Vessel, Vessel Internals and Axial Reflector Regions	Unchanged

[a]. The character in parentheses represents the computer language in which the routine is coded. A lower case "c" represents C source statements while a lower case "f" represents FORTRAN source statements. The name of FORTRAN source routines are also given in all uppercase letters.

[b]. This function calls other functions that create the appropriate cells, surfaces and material definitions for each lattice geometrical type. The models for these various lattice types are documented in individual attachments to the main document (see §8).

3. Integration Testing

Integration testing of the proper operation of this software routine can only be performed in the context of the entire process whereby lattice depletion is performed with SAS2H and a CRC model is built in MCNP.

This was accomplished by rerunning the Quad Cities Unit 1 initial core calculation utilizing BLINK, version 1 in the process. This result in comparison to the reference analysis (Ref. 7.7) is shown in Table 5-1 below.

Table 3-1 Results of Integration Testing Case

Case	Eigenvalue	Uncertainty
Reference Analysis, using MCNP 4A	1.00435	0.0004
Present Evaluation, using MCNP 4B [a], and BLINK version 1	0.99947	0.00037

[a]. The input deck generated by BLINK for this case is named, "qc1c1v_m.inp".

These results in conjunction with the results of the integration testing shown for BLINK version 0 in section 5 of Attachment VI demonstrate consistency between the reference analysis and the present evaluation using BLINK version 1.

CONTENTS

	Page
1. Introduction	5
2. Construction of MCNP Input Streams	5
2.1. QC2BOC13	5
2.2. QC2C13CP10	14
2.3. QC2C13CP11	20
2.4. QC2C13CP13	24

FIGURES

		Page
2-1	Processing of SAS2H "cut" Files by IDSGEN	6
2-2	SAS2H Fuel Assembly Identifiers for QC2BOC13	7
2-3	BLINK Fuel Assembly Identifiers for QC2BOC13	8
2-4	Full-core Control Blade Position for QC2BOC13	12
2-5	Quarter-core Control Blade Position for QC2BOC13	13
2-6	SAS2H Fuel Assembly Identifiers for QC2C13CP10	14
2-7	BLINK Fuel Assembly Identifiers for QC2C13CP10	15
2-8	Full-core Control Blade Position for QC2C13CP10	18
2-9	Quarter-core Control Blade Position for QC2C13CP10	19
2-10	Full-core Control Blade Position for QC2C13CP11	22
2-11	Quarter-core Control Blade Position for QC2C13CP11	23
2-12	Full-core Control Blade Position for QC2C13CP13	26
2-13	Quarter-core Control Blade Position for QC2C13CP13	27

TABLES

		Page
2-1	Symbolic Names for Critical Test Evaluations	5
2-2	BLINK NAMELIST Input for BOC13 CRC Reactivity Analysis	9
2-3	Locations of Nodal Length by SAS2H Fuel Assembly Identifier	10
2-4	BLINK NAMELIST Input for QC2C13CP10 CRC Reactivity Analysis	16
2-5	BLINK NAMELIST Input for QC2C13CP11 CRC Reactivity Analysis	20
2-6	BLINK NAMELIST Input for QC2C13CP13 CRC Reactivity Analysis	25

WORKSHEETS

	Page
2-1 Determination of Density for QC2BOC13	11
2-2 Determination of Density for QC2C13CP10	17
2-3 Determination of Density for QC2C13CP11	21

1. Introduction

This attachment describes the creation of the MCNP input streams to model the startup criticality tests performed on the Quad Cities Unit 2 core during cycle 13 of that core. Thus it documents the processing of Fuel Material Intermediate Datasets (FMID's) from SAS2H "cut" files and the preparation of input to BLINK, the linkage software routine used to prepare the MCNP input streams.

2. Construction of MCNP Input Streams

The documentation of this portion of the work consists of three parts:

1) identifying the relationship between BLINK fuel assemblies indices and the identifiers used for the SAS2H generation of the exposed fuel inventories (Reference 7.4);
2) specifying the various datasets used to construct the MCNP input streams and geometrical and thermodynamic parameters; and
3) defining the control blade positioning.

For cycle 13, four startup criticality tests were performed and each of three calculational parts listed above are documented for each of these distinct calculations. The symbolic notation for each of the tests modeled are shown in Table 2-1.

Table 2-1 Symbolic Names for Critical Test Evaluations

Cycle Exposure (MWd/t)	Symbolic Name
0.0	QC2BOC13
201.61	QC2C13CP10
2257.20	QC2C13CP11
6489.46	QC2C13CP13

2.1. QC2BOC13

This startup test was performed at the beginning of cycle 13 and the reactivity of the core is determined not only by the newly loaded fuel but also by the contributions of fuel assemblies loaded as far back as cycle 9. The cycle 13 core loading used is shown in Figure 2-2 (Ref. 7.9, p. 24). Note that in this figure all fresh fuel are similarly identified, since they are identical at this reactor point.

2.1.1. Correspondence of Fuel Assembly Identifiers

The relationship between the SAS2H fuel assembly identifiers and BLINK fuel assembly indices is shown in Figures 2-2 and 2-3. Note that this configuration has been rotated from the southeast quadrant to the northwest quadrant to accommodate the restrictions of the BLINK software routine. This correspondence is needed to process the SAS2H "cut" files into FMID's, since the location of the depleted fuel isotopics from SAS2H are tied to the SAS2H fuel assembly identifiers and the axial node index.

The FMID's are created by processing lists of applicable SAS2H "cut" files as shown in Figure 2-1. The SAS2H "cut" files processed in this manner have the following file name nomenclature: AaNn.dat – where "a" is the BLINK fuel assembly index and "n" is the axial node index.

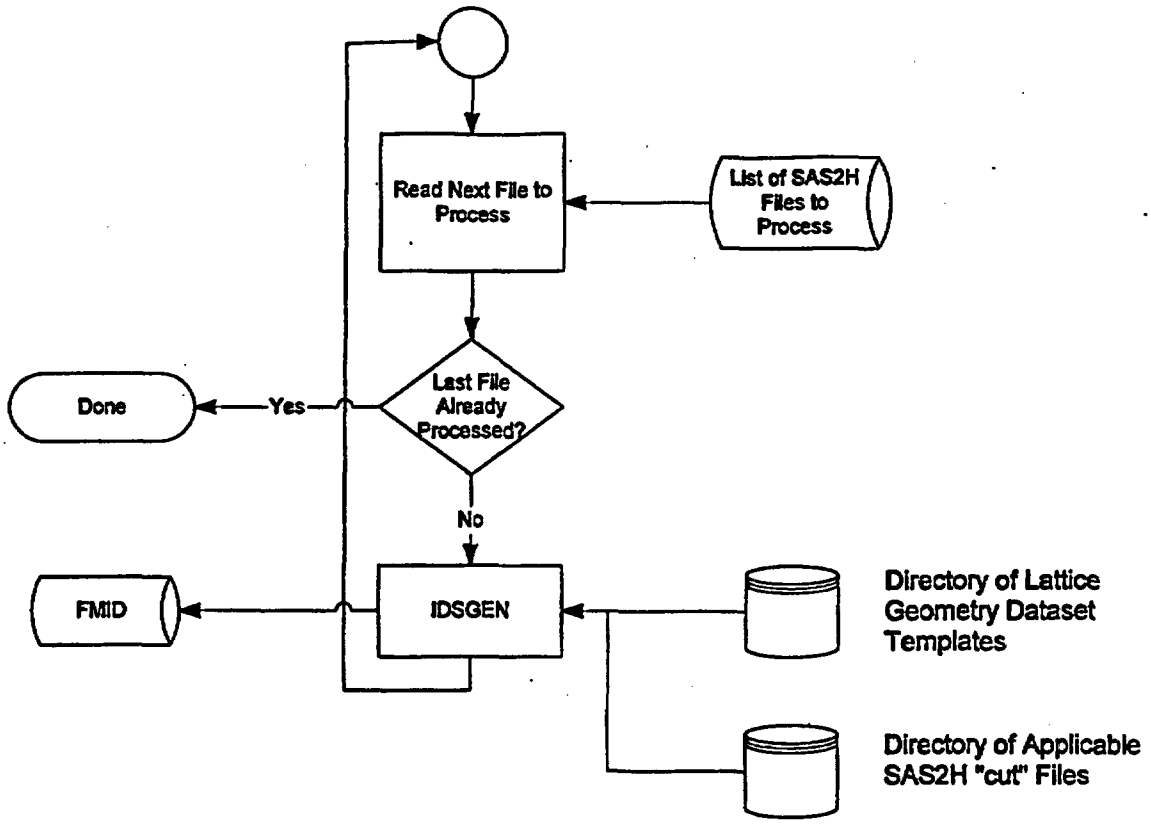

Figure 2-1 Processing of SAS2H "cut" Files by IDSGEN

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1											B4	A3	B3	B2	A4
2										A5	C5	D3	D4	D2	D8
3							A2	B5	A6	A8	D10	D6	F4*	F7*	H8
4						A1	C12	D9	C11	H9	G5	H1	G10	C8	E5
5					B1	C13	D1	D5	E14	G4	E1	E10	E3	G6	E6
6				A1	C13	A7	G11	G7	E9	H3	J1	J1	J1	E4	J1
7			A2	C12	D1	G11	E11	F1	H2	J1	H4	E12	H5	K1	F5
8			B5	D9	D5	G7	F1	E7	J1	F9	E2	F2	J1	C2	H7
9			A6	C11	E14	E9	H2	J1	C1	J1	H10	J1	H6	J1	F6
10		A5	A8	H9	G4	H3	J1	F9	J1	E13	J1	C7	J1	C3	K1
11	B4	C5	D10	G1	E1	J1	H4	E2	H10	J1	C14	F10	G8	K1	F8
12	A3	D3	D6	H11	E10	J1	E12	F2	J1	C7	F10	C9	K1	C6	G9
13	B3	D4	F4	G10	E3	J1	H5	J1	H6	J1	G8	K1	C10	J1	F3
14	B2	D2	G3	C8	G2	E4	K1	C2	J1	C3	K1	C6	J1	E8	J1
15	A4	D8	H8	E5	E6	J1	F5	H7	F6	K1	F8	G9	F3	J1	C4

NOTE: The calculations were performed with fuel assemblies identified F4 and F7 in locations (13,3) and (13,4) respectively, as shown (identified with an asterisk). The actual loading consisted of fuel assembly F7 in location (13,3) and G3 in location (13,4). This discrepancy will only have a small impact on the results, and is addressed as an assumption in the document main body.

Figure 2-2 SAS2H Fuel Assembly Identifiers for QC2BOC13

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1											12	3	11	10	4
2										5	18	30	31	29	36
3							2	13	6	8	28	36	54	57	79
4						1	14	36	14	82	65	72	61	21	41
5					9	14	28	32	37	64	37	37	39	66	42
6				1	26	7	61	67	50	74	83	83	83	40	83
7			2	25	28	71	47	51	73	83	75	37	76	84	55
8			13	35	32	67	51	43	83	60	38	52	83	15	78
9			6	24	50	45	73	83	14	83	72	83	77	83	56
10		5	8	80	64	74	83	59	83	49	83	20	83	16	84
11	12	18	36	61	37	83	75	38	81	83	27	51	68	84	60
12	3	30	33	82	46	83	48	52	83	20	60	22	84	19	71
13	11	31	54	70	39	83	76	83	77	83	68	84	23	83	53
14	10	29	63	21	62	40	84	15	83	16	84	19	83	44	83
15	4	34	79	41	42	83	55	78	56	84	58	69	53	83	17

Figure 2-3 BLINK Fuel Assembly Identifiers for QC2BOC13

2.1.2. Dataset, Geometrical and Thermodynamic Input Values

These values are those documented in Attachment VI, §3.2. The specific values selected for the FORTRAN NAMELIST portion of input for this CRC are shown in Table 2-2. The values for the assignment of fuel assembly geometrical and material types and the underlying lattice geometric and material types are consistent with the fuel assembly indices. The node lengths for each fuel assembly type are shown in Table 2-3. Note that some of these values differ slightly from those in the data source document for assembly types B, C, and D (Reference 7.9, §4.0), since the values used were based on preliminary information. The actual nodal dimensions for assembly types B through D from the data source document are the same as those shown for assembly type A in Table 2-3. These differences are small (within 1% for each affected node) and will only have a small impact the calculations. Since many of the nodes are of such great length and the fuel spacers are composed of zircaloy, which is essentially transparent to neutrons, the spacer grids are not modeled.

Table 2-2 BLINK NAMELIST Input for BOC13 CRC Reactivity Analysis

Variable	Value	Justification/Reference
CORE_DB	bwr3_724bundle.dat [a]	See Attachment II for complete description of dataset
CORE_MTLS	core_materials.dat [a]	See Attachment III for complete description of dataset
BLADE_DB	ge_d_lattice.dat [a]	See Attachment V for complete description of dataset
LPREFIX	[b]	
FPREFIX	[b]	
NAXIAL	10	This is the number of axial nodes used in the SAS2H depletion calculations.
AFL	368.91	Fuel length is fixed to be 145.24 inches (368.91 cm)
NCOLP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
NROWP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
RHO	0.969	This value is based on linear extrapolation in thermodynamic tables shown in Worksheet 2-1.
RHOBYP	0.969	The moderator density throughout the problem is assumed to be uniform, which is consistent with the startup conditions when the criticality test is performed.
TEMPK	357.5	This value corresponds to 183.8°F.
MUTP	8GUTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.
MLTP	8GLTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.

[a]. The prefix is consistent with the location where the software routine is executed.
[b]. This input is consistent with location of FMID's processed for this analysis.
[c]. See Attachment III for a complete description of this composition.

Table 2-3 Locations of Nodal Length by SAS2H Fuel Assembly Identifier

	SAS2H Fuel Assembly Identifier									
Values in inches										
Node	A	B	C	D	E	F	G	H	J	K
1	6.000	6.000	6.000	6.000	6.000	6.000	6.000	6.000	6.000	6.000
2	12.000	12.113	12.113	12.113	12.000	12.000	12.000	12.000	12.000	12.000
3	12.000	12.113	12.113	12.113	12.000	12.000	12.000	12.000	12.000	12.000
4	18.000	18.169	18.169	18.169	18.000	18.000	18.000	18.000	18.000	18.000
5	12.000	12.113	12.113	12.113	12.000	12.000	12.000	12.000	12.000	12.000
6	18.000	18.169	18.169	18.169	18.000	18.000	18.000	18.000	18.000	18.000
7	18.000	18.169	18.169	18.169	18.000	18.000	18.000	18.000	18.000	18.000
8	18.000	18.169	18.169	18.169	18.000	18.000	18.000	18.000	18.000	18.000
9	25.240	24.225	24.225	24.225	19.240	19.240	19.240	19.240	19.240	19.240
10	6.000	6.000	6.000	6.000	12.000	12.000	12.000	12.000	12.000	12.000
Values in cm										
1	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400
2	30.4800	30.7670	30.7670	30.7670	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800
3	30.4800	30.7670	30.7670	30.7670	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800
4	45.7200	46.1493	46.1493	46.1493	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200
5	30.4800	30.7670	30.7670	30.7670	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800
6	45.7200	46.1493	46.1493	46.1493	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200
7	45.7200	46.1493	46.1493	46.1493	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200
8	45.7200	46.1493	46.1493	46.1493	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200
9	64.1096	61.5315	61.5315	61.5315	48.8696	48.8696	48.8696	48.8696	48.8696	48.8696
10	15.2400	15.2400	15.2400	15.2400	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800

Worksheet 2-1 Determination of Density for QC2BOC13

Temperature (K) [a]	Pressure (Mpa)	v_f (m^3/kg)	ρ_f (g/cm^3)
273.16	0.0006113	0.001000	1.000000
275	0.000698	0.001000	1.000000
280	0.0009912	0.001000	1.000000
285	0.001388	0.001001	0.999001
290	0.001919	0.001001	0.999001
295	0.00262	0.001002	0.998004
300	0.003536	0.001004	0.996016
305	0.004718	0.001005	0.995025
310	0.00623	0.001007	0.993049
315	0.008143	0.001009	0.991080
320	0.01054	0.001011	0.989120
325	0.01353	0.001013	0.987167
330	0.01721	0.001015	0.985222
335	0.02171	0.001018	0.982318
340	0.02718	0.001021	0.979432
345	0.03377	0.001024	0.976563
350	0.04166	0.001027	0.973710
357.48	0.05347	0.001031	0.969440

[a]. These values are from Keenan and Kayes Steam Tables (Reference 7.16, hereafter cited as "Steam Tables").

2.1.3. Control Blade Positions

The control blade positions at which the reactor attained criticality are shown in Figure 2-4 (Ref. 7.9, p. 618). Since the core is modeled in quarter-core symmetry, the control blade pattern is obtained from the full-core pattern by averaging the symmetric locations. While this alters the neutron flux in the vicinity of the blades so that it is not consistent with flux distribution in any of the four locations, it is approximately correct in an integral sense inasmuch as the number of nodes covered by control blades is conserved. The blade pattern used in the MCNP analysis is shown in Figure 2-5.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1						48	0	8	0	48					
2				0	48	0	48	0	48	0	48	0			
3			0	48	0	4	0	48	0	8	0	48	0		
4		0	48	0	48	0	48	0	48	0	48	0	48	0	
5		48	0	4	0	48	0	4	0	48	0	8	0	48	
6	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
7	0	6	0	48	0	4	0	48	0	4	0	0	0	48	0
8	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
9	0	8	0	48	0	4	0	48	0	4	0	48	0	8	0
10	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
11		48	0	8	0	48	0	4	0	48	0	8	0	48	
12		0	48	0	48	0	48	0	48	0	48	0	48	0	
13			0	48	0	8	0	48	0	8	0	48	0		
14				0	48	0	48	0	48	0	48	0			
15						48	0	8	0	48					

Figure 2-4 Full-core Control Blade Position for QC2BOC13

	1	2	3	4	5	6	7	8
1						48	0	8
2				0	48	0	48	0
3			0	48	0	[illegible]	0	48
4		0	48	0	48	0	48	0
5		48	0	[illegible]	0	48	0	4
6	48	0	48	0	48	0	48	0
7	0	[illegible]	0	[illegible]	0	4	0	48
8	48	0	48	0	48	0	48	0

Figure 2-5 Quarter-core Control Blade Position for QC2BOC13

2.2. QC2C13CP10

This startup test was performed within just a few days of the beginning of cycle 13; thus while all of the fuel has been exposed, the fuel freshly loaded into cycle 13 has quite low exposure.

2.2.1. Correspondence of Fuel Assembly Identifiers

The relationship between the SAS2H fuel assembly identifiers and BLINK fuel assembly indices is shown in Figures 2-6 and 2-7. Note that again this configuration has been rotated from the southeast quadrant to the northwest quadrant to accommodate the restrictions of the BLINK software routine. These maps are different than those shown in §2.1.1 since the fuel assemblies loaded into cycle 13 are now distinct due to differences in the accumulated exposure at the location of each fuel assembly.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1											B4	A3	B3	B2	A4
2										A5	C5	D3	D4	D2	D8
3							A2	B5	A6	A8	D10	D6	F4*	F7*	H8
4						A1	C12	D9	C11	H9	G5	H1	G10	C8	E5
5					B1	C13	D1	D5	E14	G4	E1	E10	E3	G6	E6
6				A1	C13	A7	G11	G7	E9	H3	J1	J2	J3	E4	J14
7			A2	C12	D1	G11	E11	F1	H2	J15	H4	E12	H5	K1	F5
8			B5	D9	D5	G7	F1	E7	J4	F9	E2	F2	J5	C2	H7
9			A6	C11	E14	E9	H2	J4	C1	J6	H10	J12	H6	J7	F6
10		A5	A8	H9	G4	H3	J15	F9	J6	E13	J8	C7	J9	C3	K3
11	B4	C5	D10	G1	E1	J1	H4	E2	H10	J8	C14	F10	G8	K4	F8
12	A3	D3	D6	H11	E10	J2	E12	F2	J13	C7	F10	C9	K2	C6	G9
13	B3	D4	F4	G10	E3	J3	H5	J5	H6	J10	G8	K2	C10	J16	F3
14	B2	D2	G3	C8	G2	E4	K1	C2	J7	C3	K4	C6	J16	E8	J11
15	A4	D8	H8	E5	E6	J14	F5	H7	F6	K3	F8	G9	F3	J11	C4

NOTE: The calculations were performed with fuel assemblies identified F4 and F7 in locations (13,3) and (13,4) respectively, as shown (identified with an asterisk). The actual loading consisted of fuel assembly F7 in location (13,3) and G3 in location (13,4). This discrepancy will only have a small impact on the results, and is addressed as an assumption in the document main body.

Figure 2-6 SAS2H Fuel Assembly Identifiers for QC2C13CP10

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1											12	3	11	10	4
2										5	27	30	31	29	34
3							2	13	6	8	28	33	54	57	79
4						1	14	36	14	80	65	72	61	27	41
5					9	26	28	32	37	64	37	37	39	66	42
6				1	28	7	61	71	50	74	83	84	85	40	83
7			2	25	28	71	47	51	73	83	75	37	76	99	55
8			13	35	32	67	51	43	86	60	38	52	87	27	78
9			6	24	50	45	73	86	14	88	72	94	77	89	56
10		5	8	80	64	74	97	59	88	49	90	27	91	27	101
11	12	18	36	61	37	83	75	38	81	90	27	51	71	102	60
12	3	30	33	82	46	84	48	52	95	20	60	22	100	27	71
13	11	31	54	70	39	85	76	87	77	92	68	100	23	83	53
14	10	29	63	21	62	40	99	15	89	16	102	19	98	44	83
15	4	34	79	41	42	96	55	78	56	101	68	69	53	93	17

Figure 2-7 BLINK Fuel Assembly Identifiers for QC2C13CP10

2.2.2. Dataset, Geometrical and Thermodynamic Input Values

The specific values selected for the FORTRAN NAMELIST portion of input for this CRC are shown in Table 2-4. The values for the assignment of fuel assembly geometrical and material types and the underlying lattice geometric and material types are consistent with the fuel assembly indices. The node lengths for each fuel assembly type are the same as for the QC2BOC13 case (see Table 2-3). As previously, the fuel assembly spacer grids are omitted from this model.

Table 2-4 BLINK NAMELIST Input for QC2C13CP10 CRC Reactivity Analysis

Variable	Value	Justification/Reference
CORE_DB	bwr3_724bundle.dat [a]	See Attachment II for complete description of dataset
CORE_MTLS	core_materials.dat [a]	See Attachment III for complete description of dataset
BLADE_DB	ge_d_lattice.dat [a]	See Attachment V for complete description of dataset
LPREFIX	[b]	
FPREFIX	[b]	
NAXIAL	10	This is the number of axial nodes used in the SAS2H depletion calculations.
AFL	368.91	Fuel length is fixed to be 145.24 inches (368.91 cm)
NCOLP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
NROWP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
RHO	0.969	This value is based on linear extrapolation in thermodynamic tables shown in Worksheet 2-2.
RHOBYP	0.969	The moderator density throughout the problem is assumed to be uniform, which is consistent with the startup conditions when the criticality test is performed.
TEMPK	355.0	This value corresponds to 179°F.
MUTP	8GUTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.
MLTP	8GLTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.

[a]. The prefix is consistent with the location where the software routine is executed.
[b]. This input is consistent with location of FMID's processed for this analysis.
[c]. See Attachment III for a complete description of this composition.

Worksheet 2-2 Determination of Density for QC2C13CP10

Temperature (K) [a]	Pressure (MPa)	v_f (m^3/kg)	ρ_f (g/cm^3)
273.16	0.0006113	0.001000	1.000000
275	0.000698	0.001000	1.000000
280	0.0009912	0.001000	1.000000
285	0.001388	0.001001	0.999001
290	0.001919	0.001001	0.999001
295	0.00262	0.001002	0.998004
300	0.003536	0.001004	0.996016
305	0.004718	0.001005	0.995025
310	0.00623	0.001007	0.993049
315	0.008143	0.001009	0.991080
320	0.01054	0.001011	0.989120
325	0.01353	0.001013	0.987167
330	0.01721	0.001015	0.985222
335	0.02171	0.001018	0.982318
340	0.02718	0.001021	0.979432
345	0.03377	0.001024	0.976563
350	0.04166	0.001027	0.973710
354.82	0.04926	0.00103	0.97096

[a]. These values are from Steam Tables.

2.2.3. Control Blade Positions

The control blade positions at which the reactor attained criticality are shown in Figure 2-8 (Ref. 7.9, p. 618). The blade pattern used in the MCNP analysis is shown in Figure 2-9 and obtained by the same averaging scheme described in §2.1.3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1						48	0	8	0	48					
2				0	48	0	48	0	48	0	48	0			
3			0	48	0	4	0	48	0	48	0	48	0		
4		0	48	0	48	0	48	0	48	0	48	0	48	0	
5		48	0	48	0	48	0	4	0	48	0	48	0	48	
6	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
7	0	4	0	48	0	4	0	48	0	4	0	48	0	48	0
8	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
9	0	48	0	48	0	4	0	48	0	4	0	48	0	48	0
10	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
11		48	0	48	0	48	0	4	0	48	0	48	0	48	
12		0	48	0	48	0	48	0	48	0	48	0	48	0	
13			0	48	0	48	0	48	0	48	0	48	0		
14				0	48	0	48	0	48	0	48	0			
15						48	0	8	0	48					

Figure 2-8 Full-core Control Blade Position for QC2C13CP10

	1	2	3	4	5	6	7	8
1						48	0	8
2				0	48	0	48	0
3			0	48	0	[illegible]	0	48
4		0	48	0	48	0	48	0
5		48	0	[illegible]	0	48	0	4
6	48	0	48	0	48	0	48	0
7	0	[illegible]	0	48	0	4	0	48
8	48	0	48	0	48	0	48	0

Figure 2-9 Quarter-core Control Blade Position for QC2C13CP10

2.3. QC2C13CP11

This startup test was performed about a third of the way through cycle 13. All fuel assemblies have appreciable exposure.

2.3.1. Correspondence of Fuel Assembly Identifiers

The location of all fuel assemblies are the same as shown in §2.2.1.

2.3.2. Dataset, Geometrical and Thermodynamic Input Values

The specific values selected for the FORTRAN NAMELIST portion of input for this CRC are shown in Table 2-5. The values for the assignment of fuel assembly geometrical and material types and the underlying lattice geometric and material types are consistent with the fuel assembly indices. The node lengths for each fuel assembly type are the same as for the QC2BOC13 case (see Table 2-3). As previously, the fuel assembly spacer grids are omitted from this model.

Table 2-5 BLINK NAMELIST Input for QC2C13CP11 CRC Reactivity Analysis

Variable	Value	Justification/Reference
CORE_DB	bwr3_724bundle.dat [a]	See Attachment II for complete description of dataset
CORE_MTLS	core_materials.dat [a]	See Attachment III for complete description of dataset
BLADE_DB	ge_d_lattice.dat [a]	See Attachment V for complete description of dataset
LPREFIX	[b]	
FPREFIX	[b]	
NAXIAL	10	This is the number of axial nodes used in the SAS2H depletion calculations.
AFL	368.91	Fuel length is fixed to be 145.24 inches (368.91 cm)
NCOLP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
NROWP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
RHO	0.970	This value is based on linear extrapolation in thermodynamic tables shown in Worksheet 2-3.
RHOBYP	0.970	The moderator density throughout the problem is assumed to be uniform, which is consistent with the startup conditions when the criticality test is performed.
TEMPK	356.0	This value corresponds to 181.1°F.
MUTP	8GUTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.
MLTP	8GLTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.

[a]. The prefix is consistent with the location where the software routine is executed.
[b]. This input is consistent with location of FMID's processed for this analysis.
[c]. See Attachment III for a complete description of this composition.

Worksheet 2-3 Determination of Density for QC2C13CP11

Temperature (K) [a]	Pressure (MPa)	v_f (m^3/kg)	ρ_f (g/cm^3)
273.16	0.0006113	0.001000	1.000000
275	0.000698	0.001000	1.000000
280	0.0009912	0.001000	1.000000
285	0.001388	0.001001	0.999001
290	0.001919	0.001001	0.999001
295	0.00262	0.001002	0.998004
300	0.003536	0.001004	0.996016
305	0.004718	0.001005	0.995025
310	0.00623	0.001007	0.993049
315	0.008143	0.001009	0.991080
320	0.01054	0.001011	0.989120
325	0.01353	0.001013	0.987167
330	0.01721	0.001015	0.985222
335	0.02171	0.001018	0.982318
340	0.02718	0.001021	0.979432
345	0.03377	0.001024	0.976563
350	0.04166	0.001027	0.973710
355.98	0.05110	0.00103	0.970296

[a]. These values are from Steam Tables.

2.3.3. Control Blade Positions

The control blade positions at which the reactor attained criticality are shown in Figure 2-10 (Ref. 7.9, p. 619). The blade pattern used in the MCNP analysis is shown in Figure 2-11 and obtained by the same averaging scheme described in §2.1.3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1						48	0	8	0	48					
2				0	48	0	48	0	48	0	48	0			
3			0	48	0	4	0	48	0	8	0	48	0		
4		0	48	0	48	0	48	0	48	0	48	0	48	0	
5		48	0	4	0	48	0	4	0	48	0	8	0	48	
6	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
7	0	8	0	48	0	4	0	48	0	4	0	0	0	8	0
8	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
9	0	8	0	48	0	4	0	48	0	4	0	48	0	8	0
10	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
11		48	0	8	0	48	0	4	0	48	0	8	0	48	
12		0	48	0	48	0	48	0	48	0	48	0	48	0	
13			0	48	0	8	0	48	0	8	0	48	0		
14				0	48	0	48	0	48	0	48	0			
15						48	0	8	0	48					

Figure 2-10 Full-core Control Blade Position for QC2C13CP11

	1	2	3	4	5	6	7	8
1						48	0	8
2				0	48	0	48	0
3			0	48	0	[illegible]	0	48
4		0	48	0	48	0	48	0
5		48	0	[illegible]	0	48	0	4
6	48	0	48	0	48	0	48	0
7	0	8	0	[illegible]	0	4	0	48
8	48	0	48	0	48	0	48	0

Figure 2-11 Quarter-core Control Blade Position for QC2C13CP11

2.4. QC2C13CP13

This startup test was performed about two-thirds of the way through cycle 13. All fuel assemblies have appreciable exposure.

2.4.1. Correspondence of Fuel Assembly Identifiers

The location of all fuel assemblies are the same as shown in §2.2.1.

2.4.2. Dataset, Geometrical and Thermodynamic Input Values

The specific values selected for the FORTRAN NAMELIST portion of input for this CRC are shown in Table 2-6. The values for the assignment of fuel assembly geometrical and material types and the underlying lattice geometric and material types are consistent with the fuel assembly indices. The node lengths for each fuel assembly type are the same as for the QC2BOC13 case (see Table 2-3). As previously, the fuel assembly spacer grids are omitted from this model.

Table 2-6 BLINK NAMELIST Input for QC2C13CP13 CRC Reactivity Analysis

Variable	Value	Justification/Reference
CORE_DB	bwr3_724bundle.dat [a]	See Attachment II for complete description of dataset
CORE_MTLS	core_materials.dat [a]	See Attachment III for complete description of dataset
BLADE_DB	ge_d_lattice.dat [a]	See Attachment V for complete description of dataset
LPREFIX	[b]	
FPREFIX	[b]	
NAXIAL	10	This is the number of axial nodes used in the SAS2H depletion calculations.
AFL	368.91	Fuel length is fixed to be 145.24 inches (368.91 cm)
NCOLP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
NROWP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
RHO	0.970 [d]	This value is based on linear extrapolation in thermodynamic tables shown in Worksheet 2-3.
RHOBYP	0.970 [d]	The moderator density throughout the problem is assumed to be uniform, which is consistent with the startup conditions when the criticality test is performed.
TEMPK	356.0 [d]	This value corresponds to 181°F.
MUTP	8GUTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.
MLTP	8GLTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.

[a]. The prefix is consistent with the location where the software routine is executed.
[b]. This input is consistent with location of FMID's processed for this analysis.
[c]. See Attachment III for a complete description of this composition.
[d]. No reactor-averaged temperature is available for this startup test; therefore the value from QC2C13CP11 is used. The data shows relatively little variation in temperature among the startup tests and the actual temperature should be near this value.

2.4.3. Control Blade Positions

The control blade positions at which the reactor attained criticality are shown in Figure 2-12 (Ref. 7.9, p. 620). The blade pattern used in the MCNP analysis is shown in Figure 2-13 and obtained by the same averaging scheme described in §2.1.3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1						48	0	0	0	48					
2				0	48	0	48	0	48	0	48	0			
3			0	48	0	0	0	48	0	0	0	48	0		
4		0	48	0	48	0	48	0	48	0	48	0	48	0	
5		48	0	0	0	12	0	0	0	48	0	0	0	48	
6	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
7	0	0	0	12	0	0	0	12	0	0	0	48	0	0	0
8	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
9	0	0	0	22	0	0	0	12	0	0	0	48	0	0	0
10	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
11		48	0	0	0	48	0	0	0	48	0	0	0	48	
12		0	48	0	48	0	48	0	48	0	48	0	48	0	
13			0	48	0	0	0	48	0	0	0	48	0		
14				0	48	0	48	0	48	0	48	0			
15						48	0	0	0	48					

Figure 2-12 Full-core Control Blade Position for QC2C13CP13

	1	2	3	4	5	6	7	8
1						48	0	0
2				0	48	0	48	0
3			0	48	0	0	0	48
4		0	48	0	48	0	48	0
5		48	0	0	0	39	0	0
6	48	0	48	0	48	0	48	0
7	0	0	0	32	0	0	0	12
8	48	0	48	0	48	0	48	0

Figure 2-13 Quarter-core Control Blade Position for QC2C13CP13

CONTENTS

	Page
1. Introduction	5
2. Construction of MCNP Input Streams	5
2.1. QC2BOC14	5
2.2. QC214CP16	14

FIGURES

		Page
2-1	Processing of SAS2H "cut" Files by IDSGEN	6
2-2	SAS2H Fuel Assembly Identifiers for QC2BOC14	7
2-3	BLINK Fuel Assembly Identifiers for QC2BOC14	8
2-4	Full-core Control Blade Position for QC2BOC14	12
2-5	Quarter-core Control Blade Position for QC2BOC14	13
2-6	SAS2H Fuel Assembly Identifiers for QC2C14CP16	14
2-7	BLINK Fuel Assembly Identifiers for QC2C14CP16	15
2-8	Full-core Control Blade Position for QC2C14CP16	19
2-9	Quarter-core Control Blade Position for QC2C14CP16	20

TABLES

		Page
2-1	Symbolic Names for Critical Test Evaluations	5
2-2	BLINK NAMELIST Input for BOC14 CRC Reactivity Analysis	9
2-3	Locations of Nodal Length by SAS2H Fuel Assembly Identifier	10
2-4	BLINK NAMELIST Input for QC2C14CP16 CRC Reactivity Analysis	17

WORKSHEETS

	Page
2-1 Determination of Density for QC2BOC14	11
2-2 Determination of Density for QC2C14CP16	18

1. Introduction

This attachment describes the creation of the MCNP input streams to model the startup criticality tests performed on the Quad Cities Unit 2 core during cycle 14 of that core. Thus it documents the processing of Fuel Material Intermediate Datasets (FMID's) from SAS2H "cut" files and the preparation of input to BLINK, the linkage software routine used to prepare the MCNP input streams.

2. Construction of MCNP Input Streams

The documentation of this portion of the work consists of three parts:

1) Identifying the relationship between BLINK fuel assemblies indices and the identifiers used for the SAS2H generation of the exposed fuel inventories (Reference 7.4);
2) specifying the various datasets used to construct the MCNP input streams and geometrical and thermodynamic parameters; and
3) defining the control blade positioning.

For cycle 14, two startup criticality tests were performed and each of three areas will be documented for each of these distinct calculations. The symbolic notation for each of the tests modeled are shown in Table 2-1.

Table 2-1 Symbolic Names for Critical Test Evaluations

Cycle Exposure (MWd/t)	Symbolic Name
0.0	QC2BOC14
4,238.45	QC2C14CP16

2.1. QC2BOC14

This startup test was performed at the beginning of cycle 14 and the reactivity of the core is determined not only by the newly loaded fuel but also by the contributions of fuel assemblies loaded as far back as cycle 10. The cycle 14 core loading used is shown in Figure 2-2 (Ref. 7.9, p. 25). Note that in this figure all fresh fuel are similarly identified, since they are identical at this reactor point.

2.1.1. Correspondence of Fuel Assembly Identifiers

The relationship between the SAS2H fuel assembly identifiers and BLINK fuel assembly indices is shown in Figures 2-2 and 2-3. Note that this configuration has been rotated from the southeast quadrant to the northwest quadrant to accommodate the restrictions of the BLINK software routine. This correspondence is needed to process the SAS2H "cut" files into FMID's, since the location of the depleted fuel isotopics from SAS2H are tied to the SAS2H fuel assembly identifiers and the axial node index.

The FMID's are created by processing lists of applicable SAS2H "cut" files as shown in Figure 2-1. The SAS2H "cut" files processed in this manner have the following file name nomenclature: AaNn.dat – where "a" is the BLINK fuel assembly index and "n" is the axial node index.

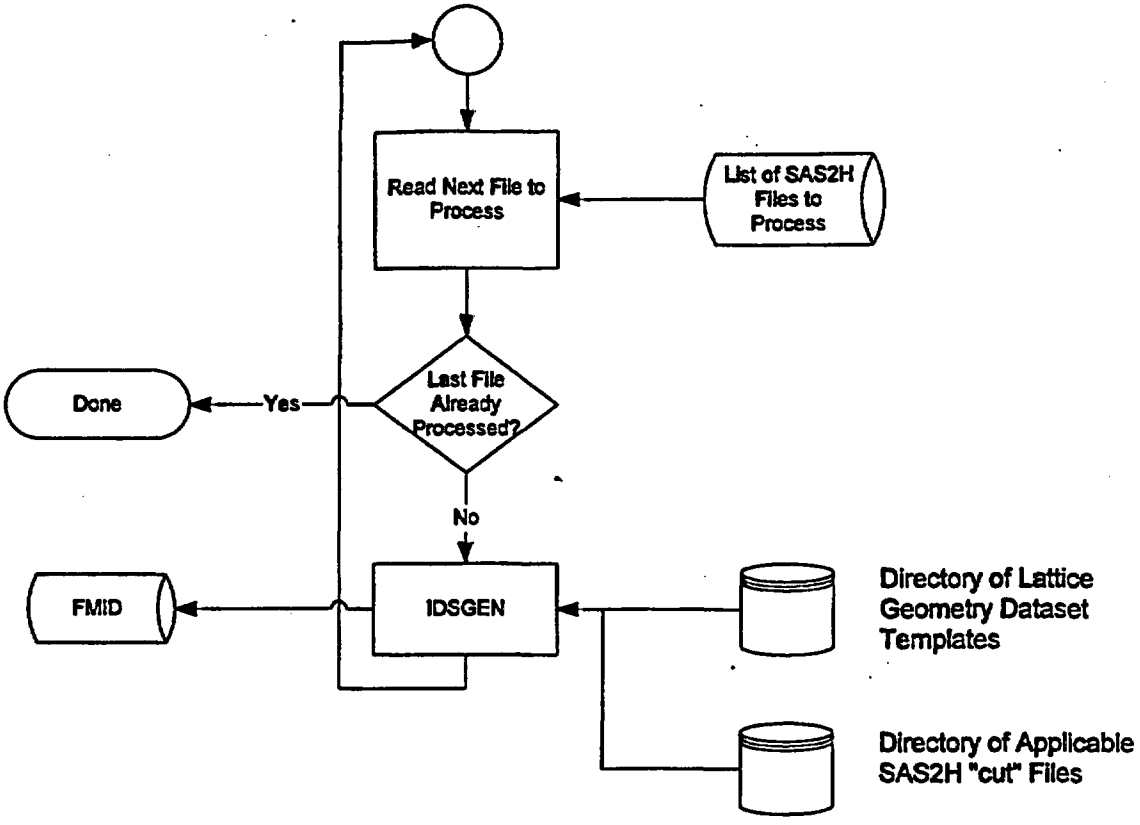

Figure 2-1 Processing of SAS2H "cut" Files by IDSGEN

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1											C14	D6	D8	D3	D5
2										C13	D4	F5	F3	E1	E2
3							C4	D10	C12	D9	F6	E10	E12	J4	J8
4						C8	F9	F2	E6	E3	J15	K4	J11	K1	G11
5					D7	C11	E4	E5	K2	J14	J1	H5	J6	G5	H7
6				C8	C11	D2	J7	J16	H4	L1	M1	M1	M1	H10	M1
7			C6	F9	E4	J7	C5	J2	K3	M1	G3	H11	G9*	M1	H2
8			D10	F2	E5	J16	J2	J13	F10	M1	E14	H9	M1	E9	J5
9			C12	E6	K2	H4	K3	F10	J12	M1	H6	L1	F8	M1	H8
10		C13	D9	E3	J14	L1	M1	M1	M1	E11	M1	G7	M1	F1	J10
11	C9	D4	F6	J15	J1	M1	G3	E14	H6	M1	G2	J3	E13	M1	H3
12	D6	F5	E10	K4	H5	M1	H1	H9	L1	G7	J3	G6	M1	G4	G10
13	D8	F3	E12	J11	J6	M1	G9*	M1	F8	M1	E8	M1	E7	M1	G9
14	D3	E1	J4	K1	G1	H10	M1	E9	M1	F1	M1	G4	M1	F7	M1
15	D5	E2	J8	G11	H7	M1	H2	J5	H8	J9	H3	G10	G9	M1	F4

NOTE: The calculations were performed with fuel assembly G9 in locations (13,7) and (7,13), as shown (identified with an asterisk). The actual loading consisted of fuel assembly G8 in these locations. This discrepancy will negligibly impact the results, and is addressed as an assumption in the document main body.

Figure 2-2 SAS2H Fuel Assembly Identifiers for QC2BOC14

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1											9	14	16	11	13
2										8	12	37	35	19	20
3							1	18	7	17	38	28	30	67	71
4						4	41	34	24	21	78	83	74	80	52
5					15	6	22	23	81	77	64	57	69	47	59
6				4	6	10	70	79	56	84	85	85	85	62	85
7			3	41	22	70	2	65	82	85	45	63	50	85	54
8			18	34	23	79	65	76	42	85	32	61	85	27	68
9			7	24	81	56	82	42	76	85	58	84	40	85	60
10		8	17	21	77	84	85	85	85	29	85	49	85	33	73
11	5	12	38	78	64	85	45	32	58	85	44	66	31	85	55
12	14	37	28	83	57	85	53	61	84	49	66	48	85	46	51
13	16	35	30	74	69	85	50	85	40	85	26	85	25	85	50
14	11	19	67	80	43	62	85	27	85	33	85	46	85	39	85
15	13	20	71	52	59	85	54	68	60	72	55	51	50	85	36

Figure 2-3 BLINK Fuel Assembly Identifiers for QC2BOC14

2.1.2. Dataset, Geometrical and Thermodynamic Input Values

These values are those documented in Attachment VI, §3.2. The specific values selected for the FORTRAN NAMELIST portion of input for this CRC are shown in Table 2-2. The values for the assignment of fuel assembly geometrical and material types and the underlying lattice geometric and material types are consistent with the fuel assembly indices. The node lengths for each fuel assembly type are shown in Table 2-3. Note that some of these values differ slightly from those in the data source document for assembly types B, C and D (Reference 7.9, §4.0), since the values used were based on preliminary information. The actual nodal dimensions for assembly types B through D from the data source document are those shown for assembly type A in Table 2-3. These differences are small (within 1% for each affected node) and will only have a small impact the calculations. Since many of the nodes are of such great length and the fuel spacers are composed of zircaloy, which is essentially transparent to neutrons, the spacer grids are not modeled.

Table 2-2 BLINK NAMELIST Input for BOC14 CRC Reactivity Analysis

Variable	Value	Justification/Reference
CORE_DB	bwr3_724bundle.dat [a]	See Attachment II for complete description of dataset
CORE_MTLS	core_materials.dat [a]	See Attachment III for complete description of dataset
BLADE_DB	ge_d_lattice.dat [a]	See Attachment V for complete description of dataset
LPREFIX	[b]	
FPREFIX	[b]	
NAXIAL	10	This is the number of axial nodes used in the SAS2H depletion calculations.
AFL	368.91	Fuel length is fixed to be 145.24 inches (368.91 cm)
NCOLP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
NROWP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
RHO	0.979	This value is based on linear extrapolation in thermodynamic tables shown in Worksheet 2-1.
RHOBYP	0.979	The moderator density throughout the problem is assumed to be uniform, which is consistent with the startup conditions when the criticality test is performed.
TEMPK	341.0	This value corresponds to 154.1°F.
MUTP	8GUTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.
MLTP	8GLTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.

[a]. The prefix is consistent with the location where the software routine is executed.
[b]. This input is consistent with location of FMID's processed for this analysis.
[c]. See Attachment III for a complete description of this composition.

Table 2-3 Locations of Nodal Length by SAS2H Fuel Assembly Identifier

	SAS2H Fuel Assembly Identifier											
Values in Inches												
Node	A	B	C	D	E	F	G	H	J	K	L	M
1	6.000	6.000	6.000	6.000	6.000	6.000	6.000	6.000	6.000	6.000	6.000	6.000
2	12.000	12.113	12.113	12.113	12.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000
3	12.000	12.113	12.113	12.113	12.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000
4	18.000	18.169	18.169	18.169	18.000	18.000	18.000	18.000	18.000	18.000	18.000	18.000
5	12.000	12.113	12.113	12.113	12.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000
6	18.000	18.169	18.169	18.169	18.000	18.000	18.000	18.000	18.000	18.000	18.000	18.000
7	18.000	18.169	18.169	18.169	18.000	18.000	18.000	18.000	18.000	18.000	18.000	18.000
8	18.000	18.169	18.169	18.169	18.000	18.000	18.000	18.000	18.000	18.000	18.000	18.000
9	25.240	24.225	24.225	24.225	19.240	19.240	19.240	19.240	19.240	19.240	19.240	19.240
10	6.000	6.000	6.000	6.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000	12.000
Values in cm												
1	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400	15.2400
2	30.4800	30.7670	30.7670	30.7670	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800
3	30.4800	30.7670	30.7670	30.7670	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800
4	45.7200	46.1493	46.1493	46.1493	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200
5	30.4800	30.7670	30.7670	30.7670	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800
6	45.7200	46.1493	46.1493	46.1493	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200
7	45.7200	46.1493	46.1493	46.1493	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200
8	45.7200	46.1493	46.1493	46.1493	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200	45.7200
9	64.1096	61.5315	61.5315	61.5315	48.8696	48.8696	48.8696	48.8696	48.8696	48.8696	48.8696	48.8696
10	15.2400	15.2400	15.2400	15.2400	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800	30.4800

Worksheet 2-1 Determination of Density for QC2BOC14

Temperature (K) [a]	Pressure (Mpa)	v_f (m^3/kg)	ρ_f (g/cm^3)
273.16	0.0006113	0.001000	1.000000
275	0.000698	0.001000	1.000000
280	0.0009912	0.001000	1.000000
285	0.001388	0.001001	0.999001
290	0.001919	0.001001	0.999001
295	0.00262	0.001002	0.998004
300	0.003536	0.001004	0.996016
305	0.004718	0.001005	0.995025
310	0.00623	0.001007	0.993049
315	0.008143	0.001009	0.991080
320	0.01054	0.001011	0.989120
325	0.01353	0.001013	0.987167
330	0.01721	0.001015	0.985222
335	0.02171	0.001018	0.982318
340	0.02718	0.001021	0.979432
340.98	0.02893	0.00102	0.978874
345	0.03377	0.001024	0.976563
350	0.04166	0.001027	0.973710

[a]. These values are from Keenan and Kayes Steam Tables (Reference 7.16, hereafter cited as "Steam Tables").

2.1.3. Control Blade Positions

The control blade positions at which the reactor attained criticality are shown in Figure 2-4 (Ref. 7.9, p. 620). Since the core is modeled in quarter-core symmetry, the control blade pattern is obtained from the full-core pattern by averaging the symmetric locations. While this alters the neutron flux in the vicinity of the blades so that it is not consistent with flux distribution in any of the four locations, it is approximately correct in an integral sense inasmuch as the number of nodes covered by control blades is conserved. The blade pattern used in the MCNP analysis is shown in Figure 2-5.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1						0	0	0	0	0					
2				0	48	0	48	0	48	0	48	0			
3			0	0	0	0	0	0	0	0	0	0	0		
4		0	48	0	48	0	48	0	48	0	48	0	48	0	
5		0	0	0	0	0	0	0	0	0	0	0	0	0	
6	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	48	0	0	0	48	0	48	0	0	0	48	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
11		0	0	0	0	0	0	0	0	0	0	0	0	0	
12		0	48	0	48	0	48	0	48	0	48	0	48	0	
13			0	0	0	0	0	0	0	0	0	0	0		
14				0	48	0	0	0	0	0	48	0			
15						0	0	0	0	0					

Figure 2-4 Full-core Control Blade Position for QC2BOC14

	1	2	3	4	5	6	7	8
1						0	0	0
2				0	48	0	16	0
3			0	0	0	0	0	0
4		0	48	0	48	0	48	0
5		0	0	0	0	0	0	0
6	48	0	48	0	48	0	48	0
7	0	0	0	0	0	0	0	0
8	0	0	48	0	0	0	48	0

Figure 2-5 Quarter-core Control Blade Position for QC2BOC14

2.2. QC214CP16

This startup test was performed at about the middle of cycle 14; thus while all of the fuel has appreciable exposure.

2.2.1. Correspondence of Fuel Assembly Identifiers

The relationship between the SAS2H fuel assembly identifiers and BLINK fuel assembly indices is shown in Figures 2-6 and 2-7. Note that again this configuration has been rotated from the southeast quadrant to the northwest quadrant to accommodate the restrictions of the BLINK software routine. These maps are different than those shown in §2.1.1 since the fuel assemblies loaded into cycle 14 are now distinct due to differences in the accumulated exposure at the location of each fuel assembly.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1											C14	D6	D8	D3	D5
2										C13	D4	F5	F3	E1	E2
3							C4	D10	C12	D9	F6	E10	E12	J4	J8
4						C8	F9	F2	E6	E3	J15	K4	J11	K1	G11
5					D7	C11	E4	E5	K2	J14	J1	H5	J6	G5	H7
6				C8	C11	D2	J7	J16	H4	L2	M13	M11	M10	H10	M2
7			C6	F9	E4	J7	C5	J2	K3	M16	G3	H11	G9*	M6	H2
8			D10	F2	E5	J16	J2	J13	F10	M15	E14	H9	M9	E9	J5
9			C12	E6	K2	H4	K3	F10	J12	M14	H6	L1	F8	M5	H8
10		C13	D9	E3	J14	L2	M16	M15	M14	E11	M12	G7	M8	F1	J10
11	C9	D4	F6	J15	J1	M13	G3	E14	H6	M12	G2	J3	E13	M4	H3
12	D6	F5	E10	K4	H5	M11	H1	H9	L1	G7	J3	G6	M7	G4	G10
13	D8	F3	E12	J11	J6	M10	G9*	M9	F8	M8	E8	M7	E7	M3	G9
14	D3	E1	J4	K1	G1	H10	M6	E9	M5	F1	M4	G4	M3	F7	M1
15	D5	E2	J8	G11	H7	M2	H2	J5	H8	J9	H3	G10	G9	M1	F4

NOTE: The calculations were performed with fuel assembly G9 in locations (13,7) and (7,13), as shown (identified with an asterisk). The actual loading consisted of fuel assembly G8 in these locations. This discrepancy will negligibly impact the results, and is addressed as an assumption in the document main body.

Figure 2-6 SAS2H Fuel Assembly Identifiers for QC2C14CP16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1											9	14	16	11	13
2										8	12	37	35	19	20
3							1	18	7	17	38	28	30	67	71
4						4	41	34	24	21	78	83	74	80	52
5					15	6	22	23	81	77	64	57	69	47	59
6				4	6	10	70	79	56	85	98	96	95	62	87
7			3	41	22	70	2	65	82	101	45	63	50	91	54
8			18	34	23	79	65	76	42	100	32	61	94	27	68
9			7	24	81	56	82	42	75	99	58	84	40	90	60
10		8	17	21	77	85	101	100	99	29	97	49	93	33	73
11	5	12	38	78	64	98	45	32	58	97	44	66	31	89	55
12	14	37	28	83	57	96	53	61	84	49	66	48	92	46	51
13	16	35	30	74	69	95	50	94	40	93	26	92	25	88	50
14	11	19	67	80	43	62	91	27	90	33	89	46	88	39	86
15	13	20	71	62	59	87	54	68	60	72	55	51	50	86	36

Figure 2-7 BLINK Fuel Assembly Identifiers for QC2C14CP16

2.2.2. Dataset, Geometrical and Thermodynamic Input Values

The specific values selected for the FORTRAN NAMELIST portion of input for this CRC are shown in Table 2-4. The values for the assignment of fuel assembly geometrical and material types and the underlying lattice geometric and material types are consistent with the fuel assembly indices. The node lengths for each fuel assembly type are the same as for the QC2BOC14 case (see Table 2-3). As previously, the fuel assembly spacer grids are omitted from this model.

Table 2-4 BLINK NAMELIST Input for QC2C14CP16 CRC Reactivity Analysis

Variable	Value	Justification/Reference
CORE_DB	bwr3_724bundle.dat [a]	See Attachment II for complete description of dataset
CORE_MTLS	core_materials.dat [a]	See Attachment III for complete description of dataset
BLADE_DB	ge_d_lattice.dat [a]	See Attachment V for complete description of dataset
LPREFIX	[b]	
FPREFIX	[b]	
NAXIAL	10	This is the number of axial nodes used in the SAS2H depletion calculations.
AFL	368.91	Fuel length is fixed to be 145.24 inches (368.91 cm)
NCOLP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
NROWP	15	Maximum dimensionality of the quarter core in units of fuel assemblies.
RHO	0.974	This value is based on linear extrapolation in thermodynamic tables shown in Worksheet 2-2.
RHOBYP	0.974	The moderator density throughout the problem is assumed to be uniform, which is consistent with the startup conditions when the criticality test is performed.
TEMPK	349.0	This value corresponds to 168.5°F.
MUTP	8GUTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.
MLTP	8GLTP0 [c]	A homogenized mixture of materials with an assumed moderator density of 1.0 g/cm^3 is used. The slight error in moderator density should have a negligible effect since this region is not in the active fuel.

[a]. The prefix is consistent with the location where the software routine is executed.
[b]. This input is consistent with location of FMID's processed for this analysis.
[c]. See Attachment III for a complete description of this composition.

Worksheet 2-2 Determination of Density for QC2C14CP16

Temperature (K) [a]	Pressure (Mpa)	v_f (m^3/kg)	ρ_f (g/cm^3)
273.16	0.0006113	0.001000	1.000000
275	0.000698	0.001000	1.000000
280	0.0009912	0.001000	1.000000
285	0.001388	0.001001	0.999001
290	0.001919	0.001001	0.999001
295	0.00262	0.001002	0.998004
300	0.003536	0.001004	0.996016
305	0.004718	0.001005	0.995025
310	0.00623	0.001007	0.993049
315	0.008143	0.001009	0.991080
320	0.01054	0.001011	0.989120
325	0.01353	0.001013	0.987167
330	0.01721	0.001015	0.985222
335	0.02171	0.001018	0.982318
340	0.02718	0.001021	0.979432
345	0.03377	0.001024	0.976563
348.98	0.04006	0.00103	0.974290
350	0.04166	0.001027	0.973710

[a]. These values are from Stream Tables.

2.2.3. Control Blade Positions

The control blade positions at which the reactor attained criticality are shown in Figure 2-8 (Ref. 7.9, p. 621). The blade pattern used in the MCNP analysis is shown in Figure 2-9 and obtained by the same averaging scheme described in §2.1.3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1						8	0	0	0	8					
2				0	48	0	48	0	48	0	48	0			
3			0	8	0	0	0	4	0	0	0	8	0		
4		0	48	0	48	0	48	0	48	0	48	0	48	0	
5		4	0	0	0	8	0	0	0	8	0	0	0	4	
6	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
7	0	0	0	8	0	0	0	4	0	0	0	8	0	0	0
8	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
9	0	0	0	8	0	0	0	4	0	0	0	8	0	0	0
10	48	0	48	0	48	0	48	0	48	0	48	0	48	0	48
11		4	0	0	0	8	0	0	0	8	0	0	0	4	
12		0	48	0	48	0	48	0	48	0	48	0	48	0	
13			0	8	0	0	0	4	0	0	0	8	0		
14				0	48	0	48	0	48	0	48	0			
15						8	0	0	0	8					

Figure 2-8 Full-core Control Blade Position for QC2C14CP16

	1	2	3	4	5	6	7	8
1						8	0	0
2				0	48	0	48	0
3			0	8	0	0	0	4
4		0	48	0	48	0	48	0
5		4	0	0	0	8	0	0
6	48	0	48	0	48	0	48	0
7	0	0	0		0	0	0	4
8	48	0	48	0	48	0	48	0

Figure 2-9 Quarter-core Control Blade Position for QC2C14CP16

CONTENTS

	Page
1. Introduction	3
2. Routines for BLINK, Version 1	4
2.1. Main Routine	4
2.2. Service Routines	23
2.3. Input Routines	44
2.4. Input Editing Routines	50
2.5. Deck Generation Routines	55

TABLES

Page

1-1 Location of Listings for Each Functional Block 3

1. Introduction

This attachment contains listings of the FORTRAN routines and C functions that changed creating version 1 of the BLINK software routine. As noted in Attachment XVII, not all the software routine components have necessarily changed, but may have been re-complied in the creation of Version 1. The listings are divided by functional block as in the description of the software routine given in Attachment XVII and Table 1-1 gives the sections in which the listing for each functional block are located.

Table 1-1 Location of Listings for Each Functional Block

Section	Contents
2.1	Main Function
2.2	Service Routines
2.3	Input Routines
2.4	Input Editing Routines
2.5	Deck Generation Routines

2. Routines for BLINK, Version 1

2.1. Main Routine

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Load Library Header Files - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Global Type Definitions - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Character Variables:
 - -   ascii_string - corresponds to record lengths for input files
 - -   ascii_record - linked list structure for loading contents of
 - -                  ascii file into memory
 - -     s_material - linked list structure for material database
*/
typedef char ascii_string[133];
typedef struct ascii_record{
                            struct ascii_record *last;
                            ascii_string line;
                            struct ascii_record *next;
                           } a_record;
typedef struct s_material{
                          struct s_material *last;
                          int atomic_number;
                          int mass_number;
                          float weight_percentage;
                          char library_suffix[5];
                          struct s_material *next;
                         } ll_material;

typedef struct u_list{
                      struct u_list *last;
                      int index;
                      ascii_string label;
                      struct u_list *next;
                     } usage_list;

typedef struct su_list{
                       struct su_list *last;
                       int index;
                       ascii_string label;
                       ascii_string value;
                       char mnemonic[4];
                       ascii_string equivalent_label;
                       struct su_list *next;
```

```
                    } surface_usage_list;

typedef struct fuel_geometry{
                              ascii_string gds_name;
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              float wrod;
                              float wrth;
                              char frcmat[6];
                              char fcmat[6];
                            } fg_list;

typedef struct all{
                    struct all *last;
                    int basis_lattice_material_index;
                    int lattice_material_index;
                    struct all *next;
                   } augmented_lattice_list;
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Function Prototypes - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -                abort - controls program termination due to error
 - -                      detection
 - -                 echo - copies input card images to output file
 - -               header - controls pagination for linkage code output
 - -                lines - tracks line numbers on current output page
 - -       memory_integer - manages memory requests for integer storage
 - -         memory_float - manages memory requests for single-precision
 - -                      real storage
 - - memory_ascii_string - manages memory requests for character storage
 - -               readin - FORTRAN routine to read input instructions
 - -              redmaps - FORTRAN routine to read two-dimensional core
 - -                      loading maps and one dimensional fuel assembly
 - -                      lattice assignment vectors
 - -                 ldlv - FORTRAN routine to read vectors of lattice
 - -                      index assignments to assembly indices
 - -              ftclose - FORTRAN routine to close logical unit
 - -               editin - prints summary of input variables and options
 - -           coredb_edit - edits contents of core geometry dataset
 - -              clattice - FORTRAN function that determines the number
 - -                      of unique lattices in the core
 - -               strngr - FORTRAN routine to read dataset names from
 - -                      FORTRAN logical unit
 - -       load_core_mtls - loads contents of core materials database
 - -                      into memory
 - - memory_ascii_record - manages memory for ascii_record structures
 - -    vessel_generation - generates input representations for core
 - -                      structural components
 - -                ccmgen - FORTRAN routine to determine and map the
 - -                      number of unique control cells
 - - core_lattice_generation -
 - -                      generate control cell lattice
 - -        generate_deck - manages the assembly of the MCNP input deck
 - -                      from the segment scratch files
 - -      copy_ascii_file - copies scratch segement files to MCNP input
 - -                      deck file
 - -       echo_MCNP_deck - echos the contents of the MCNP input deck to
 - -                      output
 - -discard_scratch_file - creates sub-process and issues command to
 - -                      delete file
 - -               memsum - summarize dynamic memory usage
*/
void abort();
void header();
void lines();
void echo(char [],char []);
int *memory_integer(int,int,int *);
float *memory_float(int,int,float *);
ascii_string *memory_ascii_string(int,int,ascii_string *);
void readin(char[],int *,char[],int *,int *,int *,float *,int *
 ,int *,int *,int *,float *,float *,float *,float *,float *,float *
 ,float *,float *,float *,char *,char *,char [],char *,char *
 ,char *,char *,char *,char *,int *,char[],char[],float *
```

```
 ,float *,float *,float *,float *);
void redmaps(int *,int *,int *,int *,int *,int *,int *,int *,int *
     ,int *,int *,int *,int *,int *,int *,int *,int *,int *, int *
     ,int *);
void redlats(int *, int *, int*, int *);
void ldlv(int *,int *,int *,int *);
void ldlvr(int *,int *,int *,float *);
void nladd(int, ascii_string *);
void ftclose(int *);
void rblade(int *,char[],int *,int *,float *,float *,float *,float *
 ,float *,float *,float *,float *,int *,float *,float *,char[],char[]
 ,char[],char[]);
void rlattice(int *,char[],int *,float *,float *,float *,float *
 ,float *,float *,float *,float *,float *, float *,float*,char[]
 ,char[],int *,int *,float *,float *);
fg_list *memory_fg_list(int,int,fg_list *);
void lodct(int *,int *,int *,int *,int *,int *,int *,int *,int *
 ,int *);
void editin(char[],char [],char [],int,int,float,int,int,int,int,int
 ,int,int *,int *,int *, int *,int *,int,int,ascii_string *
 ,ascii_string *,char [],int,char[],float,float,float,char[],char[]);
void coredb_edt(int,int,int,int,float,float,float,float,float,float
 ,float,float,float,int *,char [],char [],char [],char []
 ,char [],float,float);
void bladedb_edt(int,float,float,float,float,float,float,float,float
 ,int,float,float,char[],char[],char[],char[]);
void fgds_edt(int,char[],ascii_string *,fg_list *);
void edit_axial_nodes(int,int *,float *);
void edit_ct(int,int *p);
int clattice(int *, int *,int *,int *,int *);
void strngr(int *,int *,int *,ascii_string *);
a_record *load_core_mtls(char[],int);
a_record *memory_ascii_record(int,a_record *);
usage_list *memory_usage_list(int,usage_list *);
surface_usage_list *memory_surface_usage_list(int
 ,surface_usage_list *);
void vessel_generation(float,float,float,float,float,float,float,float
 ,float,FILE *,FILE *,FILE *,int *,int *,int *,a_record *,char[]
 ,char[],char[],char[],char[],char[],int,surface_usage_list *
 ,usage_list **,float,float);
void ccmgen(int *,int *,int *,int *,int *,int *,int *,int *
 ,int *,int *,int *,int *,int *,int *,int *);
void build_control_blade(int,float,float,float,float,float,float
 ,float,float,int,float,float,char[],char[],char[],char[],usage_list *
 ,surface_usage_list *,a_record *,FILE *,FILE *,FILE *,int *,int *
 ,float,int *,int *,int *,int *);
void generate_deck(FILE *,FILE *,FILE *,FILE *);
void copy_ascii_file(FILE *,FILE *);
void echo_MCNP_deck(FILE *);
void discard_scratch_file(char []);
void memsum();
int mchar(int *,char []);
ll_material *load_fuel_material(int *,char[],char[],int,int **,float **);
void generate_lattice_model(int *,int *,FILE *,FILE *,FILE *
 ,surface_usage_list *,usage_list *,int *,fg_list *,ascii_string *
```

```
 ,int,char[],float,float,int *,a_record *,int *,int *,char[],int *);
void build_assemblies(int *,int *,FILE *,FILE *,surface_usage_list *
 ,int **,int,int *,int,int *,int *,int *,float *,int *,int *,int,int);
usage_list *load_usage_list(char[],int,usage_list *);
void build_control_cells(int *,int *,int,int,int *,int *,int,FILE *,FILE *
 ,float,surface_usage_list *,float,int,float,float,char[],usage_list *
 ,a_record *,int *,int *,FILE *,float,float,float,int,int *,int *);
void core_lattice_generation(int *,int *,float,float,surface_usage_list *
 ,usage_list *,int *,int,int,FILE *,FILE *,int,int *,float);
void source_specification(int,float,int,int,FILE *,int,int,int *,int *
 ,float,float,int,int,int,int,int,float);
void edit_universes(int,int,int *,int *,int,int,int,int,int,int,int *
 ,int *);
void edit_surfaces(surface_usage_list *);
void editmaterials(usage_list *);
void skiprec(int *,int *);
void edit_spacer(int,int *,float *,ascii_string *);
void spacer_location(int,int *,float *,int *,int *,float *);
augmented_lattice_list *memory_lattice_list(int
 ,augmented_lattice_list *);
void augment_lattice_list(int *,int *,int *,int,int,int,int,int,int *
 ,int *,augmented_lattice_list *,int *);
ll_material *material_match(a_record *,char[],float *,int *);
void rollup_llm(ll_material *);
void fmids_check(ascii_string *,char[],int);

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Global Variables - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Integer Variables
 - -      nline - number of lines on current output page
 - -    version - version number of code
 - -  storage_i - current integer storage requested (bytes)
 - -  storage_r - current real storage requested (bytes)
 - -  storage_c - current character storage requested (bytes)
 - - storage_it - maximum integer storage requested (bytes)
 - - storage_rt - maximum real storage requested (bytes)
 - - storage_ct - maximum character storage requested (bytes)
 - -      cdate - date of code execution
 - -     crtime - time of code execution
 - -        pid - process identifier for code execution
*/
  short nline = 0, version = 1;
  int storage_i = 0, storage_r = 0, storage_it = 0, storage_rt = 0,
      storage_c = 0, storage_ct = 0;
  int long pid;
/* - Character Variables */
  char modification_level = '-';
  char codenm[8] = "BLINK";
  char cdate[9] = {NULL};
  char crtime[9] = {NULL};
  ascii_string case_title;
/* - FILE Pointers */
  FILE *nout;
```

```
void main(int argc, char *argv[]) {
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  B L I N K  *  Creates MCNP Input Decks for Modeling of
 - -                  Commercial Reactor Critical "Experiments"
 - -                  (CRC) with Fuel Constituent Number Densities
 - -                  from SAS2H Analyses
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Command Line Argument(s):
 - -   prefix - Prefix for Names of Input and Output Files
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Modification Log:
 - -   Version  Mod  Description
 - -      0      -   Initial Release
 - -      1      -   Added: Variable Noding by Geometrical Fuel Assembly
 - -                 Type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -          nin - FORTRAN logical unit number for input file
 - -      lucgeom - FORTRAN logical unit number for core geometry dataset
 - -                file
 - -  length_fin - length of input file name
 - -       naxial - number of axial nodes in fuel assemblies
 - -         nrow - number of rows in core map for the whole core
 - -         ncol - number of columns in core map for the whole core
 - -        nrowp - number of rows in core map in problem
 - -        ncolp - number of columns in the core map in the problem
 - -        nrowb - number of control blade location rows for the whole
 - -                core
 - -        ncolb - number of control blade location columns for the
 - -                whole core
 - -       nrowbp - number of control blade rows in the problem
 - -       ncolbp - number of control blade columnes in the problem
 - -      nbundlg - number of unique fuel assemblies geometrical indices
 - -      nbundlm - number of unique fuel assemblies material indices
 - -     nlatticg - number of unique fuel lattice geometrical indices
 - -     nlatticm - number of unique fuel lattice material indices
 - -        naxpl - naxial+1
 - -        valid - map of valid fuel locations in core (dynamic)
 - -         gmap - map of fuel assembly geometrical indices (dynamic)
 - -         mmap - map of fuel assembly geometrical indices (dynamic)
 - -       lgvect - vector for lattice geometrical assignment to fuel
 - -                lattices
 - -       lmvect - vector for lattice material assigment to fuel
 - -                lattices
 - -       validb - map of valid control blade locations in core (dynamic)
 - -       bladep - map of control blade axial positions (dynamic)
 - -   incore_loc - map of locations of incore instrumentation locations
 - -                (dynamic)
 - -           lt - scratch integer array used for input processing
 - -                (dynamic)
 - -        nsrck - nominal source size per cycle in MCNP
 - -          ikz - number of cycles to be skipped before beginning tally
```

```
 - -                 accumulation in MCNP
 - -          kct - number of cycles to be performed in MCNP
 - -        ncell - counter for number of cells generated for non-trans-
 - -                lated cells (this begins @ 2001 to give headroom
 - -                for indexing of cells that must be translated)
 - -     nsurface - counter for number of surfaces generated
 - -    nmaterial - counter for number of materials generated
 - -       core_f - fraction of core in problem
 - -                (1 - full, 2 - half, 4 - quarter)
 - -    nuniverse - number of universes in problem
 - -       nrowcc - number of rows in control cell map
 - -       ncolcc - number of columns in control cell map
 - -        ccmap - pointer to map of unique control cells
 - -        ntube - number of absorber tubes in control blade
 - -          ncs - number of central stiffeners in control blade wing
 - - ptr_correspondence_table
 - -              - pointer to table containing array giving correspondence
 - -                between fuel geometry indices and fuel material indices
 - -      ptr_ufl - pointer to vector of universe indices corresponding to
 - -                the lattice models created for each unique lattice type
 - -      ptr_ufa - pointer to vector of universe indices corresponding to
 - -                the fuel assembly models created for each unique
 - -                fuel assembly type
 - -          ucb - universe of control blade
 - -     fau_fill - pointer to array containing fuel assembly assignments
 - -                to control cells
 - -          ndm - periodicity for dumps to TPE file
 - -          mct - flag for creating MCTAL file
 - -                (0 - no file, <> 0 - create file)
 - -         ndmp - maximum number of dumps
 - - ptr_n_spacer
 - -              - pointer to vector containing number of spacers for
 - -                each fuel assembly geometrical type
 - - ptr_spacer_node
 - -              - pointer to array containing the axial node locations
 - -                for each fuel assembly geometrical type
 - - total_spacer_locations
 - -              - total number of spacer locations in geometrically
 - -                unique fuel assemblies
 - - nlatticm_ref
 - -              - number of lattice material indices before addition of
 - -                indices for spacer grid treatment
 - -     ncell_tr - cell indicies for translated cells
 - -   ptr_n_node - pointer to vector containing number of axial nodes
 - -                per unique fuel assembly geometrical type
 - - total_node_boundaries
 - -              - total number of node boundaries in fuel assemblies
*/
  int nin = 5, lucgeom = 1, nsrck = 10000, ikz = 10, kct = 310;
  int ndm = 5, mct = 1, ndmp = 2;
  int ncell = 2000, nsurface = 0, nmaterial = 0, core_f = 4;
  int nuniverse = 0, ncell_tr = 0;
  int ntube = 0, ncs = 0, ucb, nlatticm_ref;
  int nrowcc, ncolcc, total_spacer_locations = 0;
  int length_fin = 132;
```

```
  int naxial, nrow, ncol, nrowp, ncolp, nrowb, ncolb, nrowbp, ncolbp
    , nbundlg, nbundlm, nlatticg, nlatticm,naxpl;
  int *valid, *gmap, *mmap, *lgvect, *lmvect, *validb, *bladep, *incore_loc
  , *lt;
  int *ccmap, *ptr_n_spacer, *ptr_spacer_node;
  int *ptr_n_node;
  int *ptr_correspondence_table, *ptr_ufl, *ptr_ufa, *fau_fill;
  int total_node_boundaries = 0;
/* - Float Variable(s)
 - -                afl - active fuel length
 - -             apitch - fuel assembly pitch
 - -                sod - shroud outer radius
 - -             sthick - shroud thickness
 - -                vod - pressure vessel outer radius
 - -             vthick - pressure vessle thickness
 - -              tutpr - Top of Upper Tie Plate Region
 - -               tcgr - Top of Core Grid Region
 - -              bltpr - Bottom of Lower Tie Plate Region
 - -               bcpr - Bottom of Core Plate Region
 - -                rkk - initial guess for eigenvalue in MCNP
 - -    bypass_density - bypass water density
 - - inchannel_density - in-channel water density
 - - blade_window_offset
 - -                    - sizing of windows for control blades
 - -                      and fuel assemblies in control cell
 - -                      template
 - -             cbspan - control blade span
 - -               atid - blade absorber tube inner diameter (cm)
 - -               atod - blade absorber tube outer diameter (cm)
 - -            cbthick - blade wing thickness (cm)
 - -             trspan - blade tie rod span (cm)
 - -            trthick - blade tie rod thickness (cm)
 - -           cblength - blade active absorber length (cm)
 - -            wsthick - blade wing thickness (cm)
 - -              csoff - blade central stiffener offset (cm)
 - -            cswidth - blade central stiffener width (cm)
 - -               dtod - incore guide tube outer diameter (cm)
 - -               dtid - incore guide tube inner diameter (cm)
 - -          cb_stroke - control blade stroke (cm)
 - -                rho - input in-channel density value (g/cc)
 - -              rhobyp - input bypass density value (g/cc)
 - -               tempk - input problem temperature (Kelvins)
 - - ptr_spacer_location
 - -                    - pointer to array containing spacer
 - -                      axial location (w/r/t Bottom of Active
 - -                      Fuel)
 - - ptr_node_boundaries
 - -                    - pointer to array containing tops of
 - -                      axial nodes for each geometrically
 - -                      unique fuel assembly type
*/
  float afl, apitch, sod, sthick, vod, vthick, tutpr, tcgr, bltpr
  , bcpr, dtod, dtid, rho = 0.0, rhobyp = 0.0, tempk = 293.15;
  float cbspan = 0.0, atid = 0.0, atod = 0.0, cbthick = 0.0
   ,trspan = 0.0, trthick = 0.0, cblength = 0.0, wsthick = 0.0
```

```
   ,csoff = 0.0, cswidth = 0.0, cb_stroke = 365.76;
  float rkk = 1.0, bypass_density = 1.0, inchannel_density = 1.0;
  float blade_window_offset = 0.40;
  float *ptr_spacer_location, *ptr_node_boundaries;
/* - Character Variables
 - -          fin - name for input file
 - -         fout - name for output file
 - -       inname - FORTRAN NAMELIST on input file
 - -       prefix - prefix for names of input and output files
 - -      core_db - name of core geometry database file
 - -      lprefix - prefix for lattice geometry database
 - -      fprefix - prefix for fuel material intermediate database
 - -      lgdsnam - pointer to names of datasets for unique geometrical
 - -                lattice types
 - -      lmdsnam - pointer to names of datasets for unique geometrical
 - -                lattice types
 - -    cell_file - name for scratch file for cell definitions
 - -  surface_file - name for scratch file for surface definitions
 - - material_file - name for scratch file for material definitions
 - -    MCNP_file - name for MCNP input file
 - -    core_mtls - name of core materials database file
 - -      mvessel - vessel material identifier
 - -      mshroud - core shroud material identifier
 - -          mtg - core top guide region material identifier
 - -          mcp - core plate region material identifier
 - -         mutp - upper tie plate region material identifier
 - -         mltp - lower tie plate region material identifier
 - -         migt - incore guide tube material identifier
 - -     blade_db - dataset containing blade geometry for problem
 - -       cbpmat - identifier for blade poison material
 - -       atmat - identifier for absorber tube poison material
 - -       cbsmat - identifier for blade sheath material
 - -      cbtrmat - identifier for blade tie rod material
 - - ptr_spacer_material
 - -               - pointer to vector containing alphanumeric
 - -                 identifiers for each fuel assembly spacer
 - -                 material mnemonic
*/
  char fin[133], fout[133];
  char inname[8] = "LINKIN";
  char prefix[133], core_db[133], lprefix[133], fprefix[133]
   , cell_file[133], surface_file[133], material_file[133]
   , MCNP_file[133], core_mtls[133], blade_db[133];
  char mshroud[6], mvessel[6], mtg[6], mcp[6], mutp[6], mltp[6]
   , migt[6];
  char cbpmat[6], atmat[6], cbsmat[6], cbtrmat[6];
  ascii_string *lgdsnam, *lmdsnam, *ptr_spacer_material;
/* - file pointers
 - -   lmcnp - file for MCNP Input Deck
 - -     lu8 - scratch file for MCNP card images for cell definitions
 - -     lu9 - scratch file for MCNP card images for surface definitions
 - -    lu10 - scratch file for MCNP card images for material defini-
               tions
*/
  FILE *lmcnp, *lu8, *lu9, *lu10;
```

```
/* - Structured Variable(s)
 - -       ptr_core_mtls - pointer to first record of linked list for con-
tents
 - -                       of core materials dataset
 - -   ptr_surface_usage - pointer to first record of linked list for
 - -                       surface labels and indices
 - -  ptr_material_usage - pointer to first record of linked list for
 - -                       material labels and indices
 - -           ptr_lg_ds - pointer to lattice geometry datasets
 - - additional_lattices - material lattice indices for lattices added for
 - -                       spacer grid treatment
*/
  a_record *ptr_core_mtls;
  usage_list *ptr_material_usage;
  surface_usage_list *ptr_surface_usage;
  fg_list *ptr_lg_ds;
  augmented_lattice_list *additional_lattices;
/* - initialize global variables not explicitly initialized elsewhere */
/* - verify presence of prefix for input and output file names, if found
 - - construct files names - - - - - - - - - - - - - - - - - - - - - - - */
  if(argc < 2) {
    puts("No file prefix available from command line -- abort\n");
    return;}
  else {
    strcpy(prefix,argv[1]);
    strcpy(fin,prefix);
    strcat(fin,".inp");
    strcpy(fout,prefix);
    strcat(fout,".out");
  }
/* - open input and output files for processing */
  { FILE *inp;
    inp = fopen(fin,"r");
    if(!inp) {
      puts("Input file cannot be opened\n");
      return;}
    fclose(inp);
  }
  nout = fopen(fout,"w");
  header();
/* - echo input stream to output stream */
  echo(inname,fin);
/* - Process Input Directives - - - - - - - - - - - - - - - - - - - - - */
/* - Read Namelists */
  readin(fin,&length_fin,core_db,&nin,&lucgeom,&naxial,&afl,&nrow
   ,&ncol,&nrowp,&ncolp,&apitch,&sod,&sthick,&vod,&vthick
   ,&tutpr,&tcgr,&bltpr,&bcpr,lprefix,fprefix,core_mtls,mvessel
   ,mshroud,mtg,mcp,mutp,mltp,&core_f,blade_db,migt,&dtod,&dtid
   ,&rho,&rhobyp,&tempk);
/* - Set interal variables for in-channel and bypass water densities */
  if(rho != 0.0) inchannel_density = rho;
  if(rhobyp != 0)
    bypass_density = rhobyp;
  else
    if(rho != 0)
```

```
      bypass_density = rho;
/* - Allocate Memory for Arrays */
  nrow%2?(nrowb = nrow/2+1):(nrowb = nrow/2);
  ncol%2?(ncolb = ncol/2+1):(ncolb = ncol/2);
  nrowp%2?(nrowbp = nrowp/2+1):(nrowbp = nrowp/2);
  ncolp%2?(ncolbp = ncolp/2+1):(ncolbp = ncolp/2);
  valid = memory_integer(1,(nrow*ncol),valid);
  gmap = memory_integer(1,(nrowp*ncolp),gmap);
  mmap = memory_integer(1,(nrowp*ncolp),mmap);
  validb = memory_integer(1,(nrowb*ncolb),validb);
  bladep = memory_integer(1,(nrowbp*ncolbp),bladep);
  incore_loc = memory_integer(1,(nrowb*ncolb),incore_loc);
  lt = memory_integer(1,(nrowp*ncolp),lt);
/* - Read Arrays from Input and Core Geometry File */
  { int maxb;
    maxb = nrowp*ncolp;
    redmaps(&nin,&lucgeom,&nrow,&ncol,&nrowp,&ncolp,valid,gmap,validb
     ,bladep,&nbundlg,&maxb,lt,&nrowb,&ncolb,&nrowbp,&ncolbp
     ,incore_loc,mmap,&nbundlm);
  }
/* - Return Scratch Memory and Arrays that are No Longer Needed - - */
  valid = memory_integer(-1,(nrow*ncol),valid);
  validb = memory_integer(-1,(nrowb*ncolb),validb);
  lt = memory_integer(-1,(nrowp*ncolp),lt);
/* - Read Lattice Assignments to Fuel Types - - - - - - - - - - - - - */
  naxp1 = naxial+1;
  lgvect = memory_integer(1,(naxp1*nbundlg),lgvect);
  ldlv(&nbundlg,&naxp1,&nin,lgvect);
  lmvect = memory_integer(1,(naxp1*nbundlm),lmvect);
  ldlv(&nbundlm,&naxp1,&nin,lmvect);
/* - Determine the Number of Unique Lattices */
  { int max;
    max = naxp1*nbundlg;
    lt = memory_integer(1,max,lt);
    nlatticg = (int) clattice(&naxp1,&nbundlg,&max,lgvect,lt);
    lt = memory_integer(-1,max,lt);
    max = naxp1*nbundlm;
    lt = memory_integer(1,max,lt);
    nlatticm = (int) clattice(&naxp1,&nbundlm,&max,lmvect,lt);
    lt = memory_integer(-1,max,lt);
  }
/* - Read Names of Datasets for Unique Lattice Geometrical and - - -
 - - Material Indices */
/* - Dataset Names are Limited to 132 Characters */
  lgdsnam = memory_ascii_string(1,(nlatticg),lgdsnam);
  lmdsnam = memory_ascii_string(1,(nlatticm),lmdsnam);
/* - Dataset Names for Fuel Geometries */
  { int len = 133;
    strngr(&nin,&len,&nlatticg,lgdsnam);
/* - Dataset Names for Fuel Material Intermediate Datasets */
    strngr(&nin,&len,&nlatticm,lmdsnam);
  }
/* - Tops of Axial Nodes - - - - - - - - - - - - - - - - - - - - - - */
/* - Read Number of Axial Nodes for each Fuel Assembly */
  { int one = 1;
```

```
    ptr_n_node = memory_integer(1,nbundlg,ptr_n_node);
    ldlv(&one,&nbundlg,&nin,ptr_n_node);
  }
  { short int i;
    int *ptr_n = ptr_n_node;
    for(i = 1;i <= nbundlg;i++){
      total_node_boundaries += *ptr_n;
      ptr_n++;}
  }
/* - Read Nodal Boundaries */
  { int one = 1;
    ptr_node_boundaries = memory_float(1,total_node_boundaries
     ,ptr_node_boundaries);
    ldlvr(&one,&total_node_boundaries,&nin,ptr_node_boundaries);
  }
/* - Spacer Information - -  - - - - - - - - - - - - - - - - - - - - - - - - */
  { short int i;
    int one = 1;
/* - Read Number of Spacers for Each Fuel Assembly Geometrical
 - - Type */
    ptr_n_spacer = memory_integer(1,nbundlg,ptr_n_spacer);
    ldlv(&one,&nbundlg,&nin,ptr_n_spacer);
/* - Compute Total Number of Spacers in Problem */
    { short int i;
      int *ptr = ptr_n_spacer;
      for(i = 1;i <= nbundlg;i++){
        total_spacer_locations += *ptr;
        ptr++;}
    }
/* - Read in Spacer Locations and Material Mnemonic for each Fuel
 - - Assembly Geometrical Type */
    if(total_spacer_locations != 0){
      ptr_spacer_location = memory_float(1,total_spacer_locations
       ,ptr_spacer_location);
      ptr_spacer_material = memory_ascii_string(1,nbundlg
       ,ptr_spacer_material);
      ldlvr(&one,&total_spacer_locations,&nin,ptr_spacer_location);
      { short int i;
        int len = 133;
        strngr(&nin,&len,&nbundlg,ptr_spacer_material);
      }
    }
  }
/* - Load Contents of Fuel Geometry Datasets - - - - - - - - - - - -*/
/* - Allocate Memory */
  ptr_lg_ds = memory_fg_list(1,nlatticg,ptr_lg_ds);
  { short int i;
    int latdim, nwr;
    float cthick, asin, wgap, ngap, cradius, fsrd, cfsrd, rpitch
     ,cod, cld, pod, wrod, wrth;
    char frcmat[7], fcmat[7];
    ascii_string *as = lgdsnam, fn;
    fg_list *p;
    p = ptr_lg_ds;
      { int len=132, length;
```

```
        length = mchar(&len,lprefix);
        lprefix[length] = '\0';
        length = mchar(&len,fprefix);
        fprefix[length] = '\0';
      }
    for( i = 1; i <= nlatticg; i++,p++,as++){
      { int len=132, length;
        ascii_string holder;
        strncpy(holder,*as,(size_t) len);
        length = mchar(&len,holder);
        holder[length] = '\0';
        strncpy(*as,holder,(size_t) (length+1));
      }
      strcpy(fn,lprefix);
      strcat(fn,*as);
      rlattice(&length_fin,fn,&lucgeom,&cthick,&asin,&wgap,&ngap
       ,&cradius,&fsrd,&cfsrd,&rpitch,&cod,&cld,&pod,frcmat,fcmat
       ,&latdim,&nwr,&wrod,&wrth);
      strcpy(p->gds_name,*as);
      p->latdim = latdim; p->nwr = nwr; p->cthick = cthick;
      p->asin = asin; p->wgap = wgap; p->ngap = ngap;
      p->cradius = cradius; p->fsrd = fsrd; p->cfsrd = cfsrd;
      p->rpitch = rpitch; p->cod = cod; p->cld = cld;
      p->pod = pod; p->wrod = wrod; p->wrth = wrth;
      { int len=6, length;
        length = mchar(&len,frcmat);
        frcmat[length] = '\0';
        length = mchar(&len,fcmat);
        fcmat[length] = '\0';
      }
      strncpy(p->frcmat,frcmat,6);
      strncpy(p->fcmat,fcmat,6);}
  }
/* - Close Input File */
  ftclose(&nin);
/* - Compute Nodal Locations of Spacers */
  if(total_spacer_locations != 0){
    ptr_spacer_node = memory_integer(1,total_spacer_locations
     ,ptr_spacer_node);
    spacer_location(nbundlg,ptr_n_spacer,ptr_spacer_location
     ,ptr_spacer_node,ptr_n_node,ptr_node_boundaries);
   }
/* - Read Contents of Blade Geometry Dataset */
  rblade(&length_fin,blade_db,&lucgeom,&ntube,&cbspan,&atid,&atod
   ,&cbthick,&trspan,&trthick,&wsthick,&cblength,&ncs,&csoff,&cswidth
   ,cbpmat,atmat,cbsmat,cbtrmat);
/* - Generate Correspondence Table of Fuel Geometry Indices and Fuel
 - - Material Indices */
   ptr_correspondence_table =
    memory_integer(1,(2*nlatticm),ptr_correspondence_table);
   lodct(&nrowp,&ncolp,&naxpl,&nlatticg,&nlatticm,gmap,mmap
    ,lgvect,lmvect,ptr_correspondence_table);
/* - Edit Input Instructions - - - - - - - - - - - - - - - - - - - - - - */
  editin(core_db,lprefix,fprefix,naxial,naxpl,afl,nrowp,ncolp,nrowbp
   ,ncolbp,nbundlg,nbundlm,gmap,mmap,lgvect,lmvect,bladep,nlatticg
```

```
    ,nlatticm,lgdsnam,lmdsnam,core_mtls,core_f,blade_db,rho,rhobyp
    ,tempk,mutp,mltp);
/* - Edit Contents of Core Geometry Dataset - - - - - - - - - - - - - - - - */
  coredb_edt(nrow,ncol,nrowb,ncolb,apitch,vod,vthick,sod,sthick,tutpr
    ,tcgr,bltpr,bcpr,incore_loc,mvessel,mshroud,mtg,mcp,migt,dtod,dtid);
/* - Edit Contents of Blade Geometry Dataset - - - - - - - - - - - - - - - - */
  bladedb_edt(ntube,cbspan,atid,atod,cbthick,trspan,trthick,wsthick
    ,cblength,ncs,csoff,cswidth,cbpmat,atmat,cbsmat,cbtrmat);
/* - Edit Contents of Fuel Geometry Datasets - - - - - - - - - - - - - - - - */
  fgds_edt(nlatticg,lprefix,lgdsnam,ptr_lg_ds);
/* - Edit Node Boundaries - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  edit_axial_nodes(nbundlg,ptr_n_node,ptr_node_boundaries);
/* - Edit Spacer Input Information - - - - - - - - - - - - - - - - - - - - - - */
  if(total_spacer_locations != 0)
    edit_spacer(nbundlg,ptr_n_spacer,ptr_spacer_location
      ,ptr_spacer_material);
/* - Check for Existence of Fuel Material Intermediate Datasets - - - - - - - */
  fmids_check(lmdsnam,fprefix,nlatticm);
/* - Edit Fuel Material/Geometry Correspondence Table - - - - - - - - - - - - */
  edit_ct(nlatticm,ptr_correspondence_table);
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Add Lattice Definitions for Spacer Grid Treatment - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  nlatticm_ref = nlatticm;
  if(total_spacer_locations != 0){
    additional_lattices = memory_lattice_list(1,additional_lattices);
    augment_lattice_list(&nlatticm,lmvect,lgvect,nlatticm_ref,nlatticg
      ,nbundlm,nbundlg,naxial,ptr_n_spacer,ptr_spacer_node
      ,additional_lattices,ptr_correspondence_table);
/* - Regenerate Correspondence Table */
    ptr_correspondence_table = memory_integer(-1,(2*nlatticm_ref)
      ,ptr_correspondence_table);
    ptr_correspondence_table = memory_integer(1,(2*nlatticm)
      ,ptr_correspondence_table);
    lodct(&nrowp,&ncolp,&naxpl,&nlatticg,&nlatticm,gmap,mmap,lgvect
      ,lmvect,ptr_correspondence_table);
/* - Edit New Lattice Material Index Vector and Correspondence Table */
    lines(7);
    fprintf(nout,"0Lattice Material Index Vector has been Regenerated");
    fprintf(nout," to Accomodate Spacer Modeling\n");
    fprintf(nout,"0Fuel Assembly Index/Lattice Index\n");
    { short int i,j;
      int *p = lmvect;
      fprintf(nout,"bundle");
      for(i = 1;i <= naxial;i++)
        fprintf(nout,"%5i",i);
      fprintf(nout,"\n\n");
      for(j = 0;j < nbundlm;j++){
        fprintf(nout,"%5i ",*p);
        p++;
        for(i = 1;i < naxpl;i++){
          fprintf(nout,"%5i",*p);
          p++;
        }
        if(i > 1) lines(1);
```

```
        fprintf(nout,"\n");
      }
    }
    lines(2);
    fprintf(nout,"0New Correspondence Table\n\n");
    edit_ct(nlatticm,ptr_correspondence_table);
  }
  (void) fflush(NULL);
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Generate Input Deck for MCNP - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  strcpy(cell_file,prefix);
  strcat(cell_file,".cel");
  strcpy(surface_file,prefix);
  strcat(surface_file,".sur");
  strcpy(material_file,prefix);
  strcat(material_file,".mat");
/* - Open Scratch Files for Segments of MCNP Input Deck */
  lu8 = fopen(cell_file,"w");
  lu9 = fopen(surface_file,"w");
  lu10 = fopen(material_file,"w");
/* - Load Core Materials Database into Memory */
  ptr_core_mtls = load_core_mtls(core_mtls,length_fin);
/* - Set up first acess to surface usage pointer */
  ptr_surface_usage =
   memory_surface_usage_list(1,ptr_surface_usage);
  ptr_surface_usage->last = NULL;
  ptr_surface_usage->index = 0;
  sprintf(ptr_surface_usage->label,"\n");
  sprintf(ptr_surface_usage->value,"\n");
  sprintf(ptr_surface_usage->mnemonic,"\n");
  sprintf(ptr_surface_usage->equivalent_label,"\n");
  ptr_surface_usage->next = NULL;
/* - Write Representations for Core Structural Components */
  vessel_generation(apitch,vod,vthick,sod,sthick,tutpr,tcgr,bltpr,bcpr
   ,lu8,lu9,lu10,&ncell,&nsurface,&nmaterial,ptr_core_mtls,mvessel
   ,mshroud,mtg,mcp,mutp,mltp,core_f,ptr_surface_usage
   ,&ptr_material_usage,bypass_density,afl);
/* - Generate Control Blade Model */
  build_control_blade(ntube,cbspan,atid,atod,cbthick,trspan,trthick
   ,wsthick,cblength,ncs,csoff,cswidth,cbpmat,atmat,cbsmat,cbtrmat
   ,ptr_material_usage,ptr_surface_usage,ptr_core_mtls,lu8,lu9,lu10
   ,&nuniverse,&nmaterial,bypass_density,&ncell,&ucb,&nsurface
   ,&ncell_tr);
/* - Create Fuel Assembly Models - - - - - - - - - - - - - - - - - - - - - - */
/* - Create In-channel Moderator Material */
  { short int i, nloc;
    char *cp, zaid[11];
    ascii_string label;
    usage_list *ptr_ml = ptr_material_usage;
    do{
      ptr_ml = ptr_ml->next;}
    while(ptr_ml->next != NULL);
    strcpy(label,"Inchannel Water");
    fprintf(lu10,"c      %s\n",label);
```

```
    nmaterial++;
    ptr_ml = load_usage_list(label,nmaterial,ptr_ml);
    strcpy(zaid,"001001.50c");
    nloc = 0;
    if(nmaterial < 1000) nloc++;
    if(nmaterial < 100) nloc++;
    if(nmaterial < 10) nloc++;
    strcpy(label,"2.0");
    { short int ip;
      for( ip = 1; ip <= nloc; ip++) fprintf(lu10," ");
      fprintf(lu10,"m%i %s %s\n",nmaterial,zaid,label);
    }
    strcpy(zaid,"008016.50c");
    strcpy(label,"1.0");
    fprintf(lu10,"      %s %s\n",zaid,label);
    { short int ip;
      for( ip = 1; ip <= (nloc-1); ip++) fprintf(lu10," ");
      fprintf(lu10,"mt%i lwtr.01\n",nmaterial);
    }
  }
  ptr_ufl = memory_integer(1,nlatticm,ptr_ufl);
  { int n, ufl, nlat, index, n_entries;
    float density;
    int *p_ufl = ptr_ufl;
    ascii_string *p_sm = ptr_spacer_material;
    ll_material *ptr_mtl;
    usage_list *ptr_ml;
    augmented_lattice_list *p_addlat = additional_lattices;
    for(n = 1; n <= nlatticm_ref; n++,p_ufl++){
      generate_lattice_model(&ncell,&nmaterial,lu8,lu9,lu10
       ,ptr_surface_usage,ptr_material_usage
       ,ptr_correspondence_table,ptr_lg_ds,lmdsnam,n,fprefix
       ,inchannel_density,bypass_density,&nuniverse
       ,ptr_core_mtls,&ufl,&nsurface,"Inchannel Water",&ncell_tr);
      *p_ufl = ufl;}
/* - Add Lattices Incorporating Spacer Grids */
    if(total_spacer_locations != 0){
      for(n = (nlatticm_ref+1);n <= nlatticm;n++,p_ufl++){
/* - Determine Material Density and Add Composition to Input
 - - Deck */
        { short int i,k;
          int len = 132, length, *lgv = lgvect, nlatg, nasmg = 0;
          int *p = ptr_correspondence_table;
          ascii_string buffer;
/* - Find Lattice Geometry Index Corresponding to Lattice
 - - Material Index */
          for(i = 1;i <= nlatticm;i++)
            if(*p == n){
              p++;
              nlatg = *p;
              break;}
            else{
              p++; p++;}
          for(i = 1;i <= nbundlg;i++){
            lgv++;
```

```
            for(k = 1;k <= naxial;k++)
              if(*lgv == nlatg){
                nasmg = i;
                break;}
              else lgv++;
            if(nasmg != 0) break;
          }
          p_sm = ptr_spacer_material;
          for(i = 1;i < nasmg;i++,p_sm++);
          strncpy(buffer,*p_sm,(size_t) len);
          length = mchar(&len,*p_sm);
          buffer[length] = '\0';
          strncpy(*p_sm,buffer,(size_t) len);
        }
        ptr_mtl = material_match(ptr_core_mtls,*p_sm,&density
         ,&n_entries);
        search_usage_list(1,*p_sm,&index,ptr_material_usage);
        if(index == 0){
          nmaterial++;
          ptr_ml = ptr_material_usage;
          while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
          index = (ptr_ml->index)+1;
          ptr_ml = load_usage_list(*p_sm,index,ptr_ml);
          add_material(lu10,&nmaterial,*p_sm,n_entries,ptr_mtl);
          { short int nb,nblank = 0;
            if(nmaterial < 100) nblank = 1;
            if(nmaterial < 10) nblank = 2;
            for(nb = 1;nb <= nblank;nb++) fprintf(lu10," ");
            fprintf(lu10,"mt%i lwtr.01\n",nmaterial);
          }}
        rollup_llm(ptr_mtl);
/* - Generate New Lattices */
        nlat = p_addlat->basis_lattice_material_index;
        generate_lattice_model(&ncell,&nmaterial,lu8,lu9,lu10
         ,ptr_surface_usage,ptr_material_usage
         ,ptr_correspondence_table,ptr_lg_ds,lmdsnam,nlat
         ,fprefix,density,bypass_density,&nuniverse
         ,ptr_core_mtls,&ufl,&nsurface,*p_sm,&ncell_tr);
        *p_ufl = ufl;
        p_addlat = p_addlat->next;
      }
    }
  }
/* - Generate Unique Lattice Definitions - - - - - - - - - - - - */
  ptr_ufa = memory_integer(1,nbundlm,ptr_ufa);
  build_assemblies(&ncell,&nuniverse,lu8,lu9,ptr_surface_usage
   ,&ptr_ufa,nbundlm,lmvect,naxial,ptr_ufl,&nsurface,ptr_n_node
   ,ptr_node_boundaries,lgvect,ptr_correspondence_table,nlatticm
   ,nbundlg);
/* - Determine the Location and Number of Unique Control Cells */
  { int *ccmapw, nu_cc;
    nu_cc = nuniverse+1;
    if(core_f == 4){
      ncolcc = (ncolp/2)+1;
      nrowcc = (nrowp/2)+1;}
```

```
    if(core_f == 1){
      nrowp%4?(nrowcc = nrowp/2):(nrowcc = (nrowp/2)+1);
      ncolp%4?(ncolcc = ncolp/2):(ncolcc = (ncolp/2)+1);}
    if(core_f == 2){
      ncolcc = (ncolp/2)+1;
      nrowp%4?(nrowcc = nrowp/2):(nrowcc = (nrowp/2)+1);}
    ccmap = memory_integer(1,(ncolcc*nrowcc),ccmap);
    ccmapw = memory_integer(1,(6*ncolcc*nrowcc),ccmapw);
    ccmgen(&nrowp,&ncolp,mmap,&nrowcc,&ncolcc,ccmap,ccmapw,&core_f
     ,&ncolbp,&nrowbp,bladep,&nuniverse,incore_loc,&nrowb,&ncolb);
/* - Generate Control Cells */
    fau_fill = memory_integer(1,4*(nuniverse-nu_cc+1),fau_fill);
    build_control_cells(&ncell_tr,&nuniverse,ncolcc,nrowcc,ccmap,ccmapw
     ,(nuniverse-nu_cc+1),lu8,lu9,blade_window_offset,ptr_surface_usage
     ,apitch,ucb,cbspan,cb_stroke,migt,ptr_material_usage,ptr_core_mtls
     ,&nmaterial,ptr_ufa,lu10,bypass_density,dtid,dtod,nu_cc,fau_fill
     ,&nsurface);
/* - Genrate Control Cell Lattice */
    core_lattice_generation(&ncell,&nsurface,apitch,bypass_density
     ,ptr_surface_usage,ptr_material_usage,ccmap,ncolcc,nrowcc
     ,lu8,lu9,core_f,&nuniverse,afl);
    edit_universes(ncolcc,nrowcc,ccmap,fau_fill,core_f,ncolp,nrowp
     ,nu_cc,nlatticm,nbundlm,ptr_ufl,ptr_ufa);
    ptr_ufl = memory_integer(-1,nlatticm,ptr_ufl);
    source_specification(nsrck,rkk,ikz,kct,lu10,ncolcc,nrowcc,ccmap
     ,fau_fill,apitch,afl,core_f,ndm,mct,ndmp,(ncell-2000+ncell_tr)
     ,tempk);
    fau_fill = memory_integer(-1,4*(nuniverse-nu_cc+1),fau_fill);
    ccmapw = memory_integer(-1,(6*ncolcc*nrowcc),ccmapw);
  }
  ptr_ufa = memory_integer(-1,nbundlm,ptr_ufa);
/* - Edit Surfaces Generated */
  edit_surfaces(ptr_surface_usage);
  edit_materials(ptr_material_usage);
/* - Close Scratch File for Deck Segments */
  fclose(lu8);
  fclose(lu9);
  fclose(lu10);
/* - Build MCNP Deck from Segment Files */
  strcpy(MCNP_file,prefix);
  strcat(MCNP_file,"_m.inp");
  lmcnp = fopen(MCNP_file,"w");
/* - Open Scratch Files for Segments of MCNP Input Deck */
  lu8 = fopen(cell_file,"r");
  lu9 = fopen(surface_file,"r");
  lu10 = fopen(material_file,"r");
  generate_deck(lmcnp,lu8,lu9,lu10);
  fclose(lu8); fclose(lu9); fclose(lu10);
  fclose(lmcnp);
/* - Discard Scratch Files */
  discard_scratch_file(cell_file);
  discard_scratch_file(surface_file);
  discard_scratch_file(material_file);
/* - Echo MCNP Input Deck to Output File */
  lmcnp = fopen(MCNP_file,"r");
```

```
  echo_MCNP_deck(lmcnp);
/* - Return Memory */
  incore_loc = memory_integer(-1,(nrowb*ncolb),incore_loc);
  bladep = memory_integer(-1,(nrowbp*ncolbp),bladep);
  gmap = memory_integer(-1,(nrowp*ncolp),gmap);
  mmap = memory_integer(-1,(nrowp*ncolp),mmap);
  lgvect = memory_integer(-1,(naxpl*nbundlg),lgvect);
  lmvect = memory_integer(-1,(naxpl*nbundlm),lmvect);
  lgdsnam = memory_ascii_string(-1,(nlatticg),lgdsnam);
  lmdsnam = memory_ascii_string(-1,(nlatticm_ref),lmdsnam);
  ptr_lg_ds = memory_fg_list(-1,nlatticg,ptr_lg_ds);
   ptr_correspondence_table =
    memory_integer(-1,(2*nlatticm),ptr_correspondence_table);
/* - end of processing - - - - - - - - - - - - - - - - - - - - - - - - - - */
  memsum();
  fclose(nout);
  return;
}
```

2.2. Service Routines

Function load_core_mtls

```
#include<stdio.h>
#include<string.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;

a_record *memory_ascii_record(int,int,a_record *);


a_record *load_core_mtls(char core_mtls[],int length_fin){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * l o a d _ c o r e _ m t l s  *  Loads Core Materials Dataset
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -  core_mtls - specification for file containing core mat-  (input)
 - -              materials definitions
 - - length_fin - length of file name                          (input)
 - -      first - pointer to first element in linked list     (output)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - -  Integer Variable(s)
 - -        c - single character used to process sucessive characters
 - -            from the input file
*/
  unsigned char c;
/* -  Character Variable(s)
 - -   record - string used to hold single lines from the input file
 - -        p - pointer to location in record
*/
  ascii_string record;
  char *p;
/* -  Structured Variable(s)
 - -        first - pointer to first member of linked list
 - -  ptr_current - pointer to current member of linked list
 - -     ptr_next - pointer to next member of linked list
*/
  a_record *first, *ptr_current, *ptr_next;
/* - FILE Variable(s)
      lu - pointer to core materials dataset
*/
  FILE *lu;
/* - Open Core Materials Dataset - - - - - - - - - - - - - - - - - - - - - */
/*  { short int i;
    for(i = 0;i < length_fin;i++){
      if(core_mtls[i] == ' '){
        core_mtls[i] = '\0';
```

```
        break;}
      }
    } */
    { int len = 132, length;
      length = mchar(&len,core_mtls);
      core_mtls[length] = '\0';
    }
    lu = fopen(core_mtls,"r");
/* - Allocate First Structure for Processing File */
    ptr_current = memory_ascii_record(1,1,ptr_current);
    first = ptr_current;
    ptr_current->last = NULL;
    p = record;
    while((c = fgetc(lu)) != 10){
      *p = c;
      p++;}
    sprintf(p,"\n");
    strcpy(ptr_current->line,record);
    ptr_current->next = NULL;
/* - read balance of records in dataset */
    p = record;
    while((c = fgetc(lu)) != 255){
      *p = c;
      p++;
      if(c == 10){
        sprintf(p,"\n");
        ptr_next = memory_ascii_record(1,1,ptr_next);
        ptr_next->next = NULL;
        ptr_current->next = ptr_next;
        ptr_next->last = ptr_current;
        ptr_current = ptr_next;
        strcpy(ptr_current->line,record);
        p = record;}
    }
    return first;
}
```

Function load_fuel_material

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;


ll_material *memory_s_material(int,int,ll_material *);
float *memory_float(int,int,float *);

ll_material *load_fuel_material(int *nft,char fprefix[],char dataset[]
```

```
 ,int nlat,int **lattice,float **fp_density){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * l o a d _ f u e l _ m a t e r i a l *  Loads Contents of Fuel
 - -                                          Material Dataset into
 - -                                          Memory for Subsequent
 - -                                          Processing
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -        nft - Number of Fuel Rod Types in Lattice          (output)
 - -    fprefix - Prefix for Lattice Composition Database       (input)
 - -    dataset - Dataset Name for Fuel Intermediate Dataset of  (input)
 - -              Interest
 - -       nlat - lattice dimensionality                        (input)
 - -    lattice - Fuel Rod Type Map                             (output)
 - -          p - Vector of Pointers to Start of Linked List    (output)
 - -              Containing Material Inventories for each
 - -              Distinct Fuel Rod Type
 - - fp_density - densities for each fuel type                  (output)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  n - loop index
*/
  short int n;
/* - Character Variable(s)
 - -       c - single character for reading input file
 - -  buffer - string variable used to manage dataset processing
 - -      pb - pointer to buffer
 - -      fn - complete dataset file descriptor
 - -  header - header from dataset
*/
  unsigned char c;
  char *pb;
  ascii_string fn, buffer, header;
/* - File Variable(s)
 - -   ds - dataset stream pointer
 - - nout - output file
*/
  FILE *ds;
  extern FILE *nout;
/* - Structured Variable(s)
*/
  ll_material *p, *p_return;
/* - Put Dataset Name in C String Format - - - - - - - - - - - - - - - - - - */
  { int len = 132, length;
    length = mchar(&len,dataset);
    dataset[length] = '\0';
  }
  strcpy(fn,fprefix);
  strcat(fn,dataset);
/* - Open File for Processing - - - - - - - - - - - - - - - - - - - - - - - - */
  ds = fopen(fn,"r");
  if(!ds){
    lines(5);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout
     ," Load Fuel Material, Fuel Material Dataset not Found\n");
    fprintf(nout,"0Dataset Name:\n");
    fprintf(nout," %s\n",fn);
    abort();}
/* - Read Header Record */
  strcpy(buffer,"");
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while (c != 10);
  *pb = '\0';
  strcpy(header,buffer);
/* - Skip QA Record */
  do
    c = fgetc(ds);
  while (c != 10);
/* - Skip Fuel Rod Type Map Title */
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while (c != 10);
/* - Read Fuel Rod Type Map Line by Line */
  { short int j;
    int *plattice;
    ascii_string holder;
    plattice = *lattice;
    for( j = 1; j <= nlat; j++){
      pb = buffer;
      do{
        c = fgetc(ds);
        *pb = c;
        pb++;}
      while(c != 10);
      *pb = '\0';
      pb = buffer;
      for(n = 0; n < nlat; n++,plattice++){
        while(*pb == ' ') pb++;
        strcpy(holder,"");
        do{
          strncat(holder,pb,1);
          pb++;}
        while(*pb != ' ');
        pb++;
        sscanf(holder,"%i",plattice);}
    }
/* - Compute Number of Lattice Types */
    *nft = 0;
    plattice = *lattice;
    for( j = 1; j <= (nlat*nlat); j++,plattice++)
```

```
      if(*plattice > *nft) *nft = *plattice;
    p = memory_s_material(1,*nft,p);
  }
/* - Skip Title for Density Vector */
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while (c != 10);
/* - Allocate Memory for Density Vector */
  *fp_density = memory_float(1,*nft,*fp_density);
/* - Read Density Values */
  { short int nmax,i;
    ascii_string holder;
    float *p = *fp_density;
    n = 1;
    do{
      pb = buffer;
      do{
        c = fgetc(ds);
        *pb = c;
        pb++;}
      while(c != 10);
      *pb = '\0';
      pb = buffer;
      if((n+5) > *nft) nmax = *nft;
      else nmax = n+4;
      for( i = n; i <= nmax; i++,p++){
        while(*pb == ',') pb++;
        strcpy(holder,"");
        c = *pb;
        while((*pb != ',') && (c != 10)){
          strncat(holder,pb,1);
          pb++;
          c = *pb;}
        pb++;
        sscanf(holder,"%f",p);}
      n = nmax+1;
    }
    while(n <= *nft);
  }
/* - Skip Title for Fuel Material Compositions */
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while (c != 10);
/* - Read Each Materials for Each Fuel Rod Type */
/* - Allocate Memory for Vector of Linked Lists */
  p = memory_s_material(1,*nft,p);
  p_return = p;
  { unsigned char c;
    int index, n_entries, i, j;
```

```
char label[6];
ascii_string holder;
ll_material *ptr_mtl, *ptr_last, *ptr_next;
for( n = 0; n < *nft; n++){
  ptr_mtl = p;
  ptr_last = NULL;
  ptr_mtl->atomic_number = 0;
  strcpy(ptr_mtl->library_suffix,"");
  ptr_mtl->mass_number = 0;
  ptr_mtl->weight_percentage = 0.0;
  pb = buffer;
  do{
    c = fgetc(ds);
    *pb = c;
    pb++;}
  while(c != 10);
  *pb = '\0';
  if(*nft > 1)
    sscanf(buffer,"%s %i %i",label,&index,&n_entries);
  else
    sscanf(buffer,"%i",&n_entries);
  for( i = 0; i < n_entries; i++){
    pb = buffer;
    do{
      c = fgetc(ds);
      *pb = c;
      pb++;}
    while(c != 10);
    *pb = '\0';
    pb = buffer;
    c = *pb;
    for( j = 1; j < 5; j++){
      strcpy(holder,"");
      while((*pb != ',') && (c != 10)){
        strncat(holder,pb,1);
        pb++;
        c = *pb;}
      switch (j) {
        case 1:
          { int value;
            sscanf(holder,"%i",&value);
            ptr_mtl->atomic_number = value;
            pb++;
          }
          break;
        case 2:
          { ascii_string value;
            sscanf(holder,"%s",value);
            strncpy(ptr_mtl->library_suffix,value,4);
            pb++;
          }
          break;
        case 3:
          { int value;
            sscanf(holder,"%i",&value);
```

```
                ptr_mtl->mass_number = value;
                pb++;
              }
              break;
            case 4:
              { float value;
                sscanf(holder,"%f",&value);
                ptr_mtl->weight_percentage = value;
                pb++;
              }
          }
        }
        ptr_mtl->next = NULL;
        ptr_mtl->last = ptr_last;
        ptr_last = ptr_mtl;
        if( i < (n_entries-1)){
          ptr_next = memory_s_material(1,1,ptr_next);
          ptr_mtl->next = ptr_next;
          ptr_mtl = ptr_next;
          ptr_mtl->atomic_number = 0;
          strcpy(ptr_mtl->library_suffix,"");
          ptr_mtl->mass_number = 0;
          ptr_mtl->weight_percentage = 0.0;
          }
      }
    p++;
    }
  }
  fclose(ds);
  return p_return;
}
```

Function load_surface_usage_list

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;

surface_usage_list *memory_surface_usage_list(int,surface_usage_list *);

surface_usage_list *load_surface_usage_list(ascii_string label,int index
 ,ascii_string value,char mnemonic[4],ascii_string equivalent_label
 ,surface_usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - load_surface_usage_list - loads new values into surface usage list
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      label - character string to be added to new structure  (input)
 - -      index - index for character string                    (input)
 - -      value - definition of surface                         (input)
 - -   mnemonic - surface definition mnemonic                   (input)
 - - equivalent_label
 - -            - label of
 - -          p - pointer to memory locations                     (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Structured Variable(s)
 - -  p_new - pointer to new structure
*/
  surface_usage_list *p_new;
/* - Check for Initial Member of String - - - - - - - - - - - - - - - - - - */
  if(p == NULL){
/* - Initial Member */
    p = memory_surface_usage_list(1,p);
    p->last = NULL;
    p->next = NULL;
    strcpy(p->label,label);
    strcpy(p->value,value);
    strcpy(p->mnemonic,mnemonic);
    strcpy(p->equivalent_label,equivalent_label);
    p->index = index;
    return p;}
/* - Addition to Linked List */
  else{
    p_new = memory_surface_usage_list(1,p_new);
    p->next = p_new;
    p_new->last = p;
    p = p_new;
    p->next = NULL;
    strcpy(p->label,label);
    strcpy(p->value,value);
    strcpy(p->mnemonic,mnemonic);
    strcpy(p->equivalent_label,equivalent_label);
    p->index = index;
    return p;}
}
```

Function memory_ascii_record

```
#include<stdio.h>

#include<malloc.h>
#include<stdlib.h>
#include<errno.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
```

```
                        } a_record;

a_record *memory_ascii_record(int operation,int number, a_record *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_ascii_record - manages memory requests for character storage
 - -                       for variables of the ascii_record structure
 - -                       type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform          (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures for which to allocate     (input)
 - -             memory
 - -         p - pointer to memory locations                     (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = number*sizeof(a_record);
    p = (a_record *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_ascii_record: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_ascii_record: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += number*sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    return p;}
  else {
    size = number*sizeof(ascii_string);
    storage_c -= size;
```

```
    free((void *) p);
    p = NULL;
    return p;}
}
```

Function memory_fg_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct fuel_geometry{
                              ascii_string gds_name;
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              float wrod;
                              float wrth;
                              char frcmat[6];
                              char fcmat[6];
                            } fg_list;

fg_list *memory_fg_list(int operation, int number, fg_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_fg_list - manages memory requests for storage for
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures to allocate                (input)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_r - current bytes of single-precision real storage
 - -              requested
 - -  storage_rt - maximum bytes of single-precision real storage
```

```
 - -                 requested
*/
  extern int storage_i, storage_it;
  extern int storage_r, storage_rt;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    p = (fg_list *)calloc((size_t) number,(size_t) sizeof(fg_list));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_fg_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_fg_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += 2*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    storage_r += 12*sizeof(float);
    if(storage_r > storage_rt) storage_rt = storage_r;
    storage_c += 12*sizeof(char)+sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_i -= 2*sizeof(int);
    storage_r -= 12*sizeof(float);
    storage_c -= 12*sizeof(char)+sizeof(ascii_string);
    return p;}
}
```

Function memory_float

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>

float *memory_float(int operation, int number, float *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_integer - manages memory requests for single-precision real
 - -                  storage
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
```

```
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of storage locations to request          (input)
 - -             (n.b., not used for return requests)
 - -         p - pointer to memory location                       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   storage_r - current bytes of float storage requested
 - -  storage_rt - maximum bytes of float storage requested
*/
  extern int storage_r, storage_rt;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    p = (float *)calloc((size_t)number,(size_t) sizeof(float));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_float: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_float: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_r += number;
    if(storage_r > storage_rt) storage_rt = storage_r;
    return p;}
  else {
    storage_r -= number;
    free((void *) p);
    p = NULL;
    return p;};
}
```

Function memory_integer

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>

int *memory_integer(int operation, int number, int *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_integer - manages memory requests for integer storage
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of storage locations to request          (input)
 - -             (n.b., not used for return requests)
 - -         p - pointer to memory location                       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
*/
  extern int storage_i, storage_it;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    p = (int *)calloc((size_t)number,(size_t) sizeof(int));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_integer: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
        ,"0***  F A T A L  E R R O R  *** -- Function memory_integer: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += number;
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    storage_i -= number;
    free((void *) p);
    p = NULL;
    return p;};
}
```

Function memory_lattice_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef struct all{
```

```
                    struct all *last;
                    int basis_lattice_material_index;
                    int lattice_material_index;
                    struct all *next;
                   } augmented_lattice_list;

augmented_lattice_list *memory_lattice_list(int operation
 ,augmented_lattice_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_lattice_list - manages memory requests for storage for vari-
 - -                       ables of the augmented_lattice_list structure
 - -                       type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(augmented_lattice_list);
    p = (augmented_lattice_list *)calloc((size_t) 1
     ,(size_t) sizeof(augmented_lattice_list));
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_lattice_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_lattice_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_i += 2*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
```

```
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_i -= 2*sizeof(int);
    return p;}
}
```

Function memory_s_material

```
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
#include<errno.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;


ll_material *memory_s_material(int operation,int number,ll_material *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_s_material - manages memory requests for storage for vari-
 - -                     ables of the s_material structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -    number - number of structures to process                 (input)
 - -         p - pointer to memory locations                       (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_r - current bytes of single-precision real storage
 - -              requested
 - -  storage_rt - maximum bytes of single-precision real storage
 - -              requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
```

```
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_r, storage_rt;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = number*sizeof(ll_material);
    p = (ll_material *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_s_material: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_s_material: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += 3*number*sizeof(char);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += 2*number*sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    storage_r += number*sizeof(float);
    if(storage_r > storage_rt) storage_rt = storage_r;
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_c -= 3*number*sizeof(char);
    storage_i -= 2*number*sizeof(int);
    storage_r -= number*sizeof(float);
    return p;}
}
```

Function memory_surface_usage_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
```

```
                    ascii_string label;
                    ascii_string value;
                    char mnemonic[4];
                    ascii_string equivalent_label;
                    struct su_list *next;
                  } surface_usage_list;

surface_usage_list *memory_surface_usage_list(int operation
 , surface_usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_surface_usage_list - manages memory requests for storage for
 - -                             variables of the surface_usage_list
 - -                             structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(surface_usage_list);
    p = (surface_usage_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Error Number %i Detected\n",errno);
      }
```

```
      lines(2);
      abort();
    }
    storage_c += 3*sizeof(ascii_string)+4*sizeof(char);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_c -= (3*sizeof(ascii_string)+4*sizeof(char));
    storage_i -= sizeof(int);
    return p;}
}
```

Function memory_usage_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

usage_list *memory_usage_list(int operation, usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_usage_list - manages memory requests for storage for vari-
 - -                     ables of the usage_list structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_c, storage_ct;
```

```
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(usage_list);
    p = (usage_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_usage_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function memory_usage_list: ");
        fprintf(nout,"Error Number %i Detected\n");
      }
      lines(2);
      abort();
    }
    storage_c += sizeof(ascii_string);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
  else {
    free((void *) p);
    p = NULL;
    storage_c -= sizeof(ascii_string);
    storage_i -= sizeof(int);
    return p;}
}
```

Function search_surface_usage_list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <errno.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;

surface_usage_list *memory_surface_usage_list(int operation
```

```
 , surface_usage_list *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - memory_surface_usage_list - manages memory requests for storage for
 - -                             variables of the surface_usage_list
 - -                             structure type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -             ( 1 - allocate memory,
 - -              -1 - return memory)
 - -         p - pointer to memory locations                      (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -           n - number of entries in proper size
 - -        size - number of bytes requested
 - -   storage_i - current bytes of integer storage requested
 - -  storage_it - maximum bytes of integer storage requested
 - -   storage_c - current bytes of character storage requested
 - -  storage_ct - maximum bytes of character storage requested
*/
  size_t size ;
  extern int storage_i, storage_it;
  extern int storage_c, storage_ct;
/* - FILE Pointer Variable(s)
 - -  nout - output file */
  extern FILE *nout;
/* - - Process Memory Request - - - - - - - - - - - - - - - - - - - - - - - - */
  if(operation == 1) {
    size = sizeof(surface_usage_list);
    p = (surface_usage_list *)malloc(size);
    if((errno != NULL) || (p == NULL)) {
      if(p == NULL) {
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Null Pointer Returned\n");
      }
      if(errno != NULL){
        lines(2);
        fprintf(nout
     ,"0***  F A T A L  E R R O R  *** -- Function");
        fprintf(nout," memory_surface_usage_list: ");
        fprintf(nout,"Error Number %i Detected\n",errno);
      }
      lines(2);
      abort();
    }
    storage_c += 3*sizeof(ascii_string)+4*sizeof(char);
    if(storage_c > storage_ct) storage_ct = storage_c;
    storage_i += sizeof(int);
    if(storage_i > storage_it) storage_it = storage_i;
    return p;}
```

```
  else {
    free((void *) p);
    p = NULL;
    storage_c -= (3*sizeof(ascii_string)+4*sizeof(char));
    storage_i -= sizeof(int);
    return p;}
}
```

2.3. Input Routines

Subroutine ldlv

```
      SUBROUTINE LDLV(NBUNDLE,NAXP1,LU,LVECT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  L D L V *  Reads Vectors giving Lattice Assignments to Fuel
C - -                 Assemblies
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -      NBUNDLE - Number of Fuel Assemblies in Core          (input)
C - -        NAXP1 - Number of Axial Nodes in Core +1           (input)
C - -           LU - Logical Unit from which to Read Vectors    (input)
C - -        LVECT - Lattice Loading Vectors for each Fuel     (output)
C - -                Assembly
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MESSAGE - Character String for Messages
      CHARACTER*132 MESSAGE
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION LVECT(NAXP1,NBUNDLE)
C - - Skip Header Record - - - - - - - - - - - - - - - - - - - - - - - -
      READ(LU,'(1X)')
      DO NB=1,NBUNDLE
        READ(LU,*,END=500)  (LVECT(I,NB),I=1,NAXP1)
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
      RETURN
C - - Error Processing - - - - - - - - - - - - - - - - - - - - - - - - -
  500 MESSAGE = '0***  F A T A L  E R R O R  ***  SUBR. LDLV --'
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' Premature End-of-File Encountered on Logical Unit'')')
      WRITE(MESSAGE((MCHAR(132,MESSAGE)+1):)
     2 ,'('' '',I2)') LU
      CALL lines(5)
      CALL fortran_message(MESSAGE)
      CALL abort()
      END
```

Subroutine lodct

```
      SUBROUTINE LODCT(NCOLP,NROWP,NAXP1,NLATTICG,NLATTICM,GMAP,MMAP
     2 ,LGVECT,LMVECT,CTABLE)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   *  L O D C T *  Load Correspondence Table Matching Lattice
C - -                  Geometry Types with Lattice Material Types
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        NCOLP - Number of Columns in Core Fuel Assembly    (input)
C - -                Map
C - -        NROWP - Number of Rows in Core Fuel Assembly Map   (input)
C - -        NAXP1 - Number of Axial Nodes + 1                  (input)
C - -     NLATTICG - Number of Unique Lattice Geometry Types    (input)
C - -     NLATTICM - Number of Unique Lattice Material Types    (input)
C - -         GMAP - Fuel Assembly Geometry Index Map           (input)
```

```
C - -          MMAP - Fuel Assembly Material Index Map            (input)
C - -        LGVECT - Vectors Provides Lattice Geometry Assign-   (input)
C - -                 ment Indices for a Given Fuel Assembly
C - -                 Geometry Index
C - -        LMVECT - Vectors Provides Lattice Material Assign-   (input)
C - -                 ment Indices for a Given Fuel Assembly
C - -                 Material Index
C - -        CTABLE - Correspondence Table Providing Lattice     (output)
C - -                 Geometry Index for a Given Lattice Mater-
C - -                 ial Index
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -  MATCH - Flag to Indicate that a Lattice Material/Lattice
C - -           Geometry Pair has Already been Added to CTABLE
      LOGICAL MATCH
      INTEGER GMAP(NCOLP,NROWP),CTABLE(2,NLATTICM)
C - - Dimension Statement(s) - - - - - - - - - - - - - - - - - - - - - - -
      DIMENSION MMAP(NCOLP,NROWP),LGVECT(NAXP1,NLATTICG)
     2 ,LMVECT(NAXP1,NLATTICM)
C - - Sweep Over Entire Core - - - - - - - - - - - - - - - - - - - - - - -
      CALL INVALI((2*NLATTICM),CTABLE,0)
      IC = 0
      DO J = 1,NROWP
        DO I = 1,NCOLP
          IF(MMAP(I,J) .NE. 0) THEN
C - - Examine Each Node of Fuel Assembly Loading Vector
            DO K = 2,NAXP1
              LGV = LGVECT(K,GMAP(I,J))
              LMV = LMVECT(K,MMAP(I,J))
              IF(IC .EQ. 0) THEN
                IC = IC+1
                CTABLE(1,IC) = LMV
                CTABLE(2,IC) = LGV
              ELSE
                MATCH = .FALSE.
                DO N = 1,IC
                  IF((CTABLE(1,N) .EQ. LMV) .AND.
     2                (CTABLE(2,N) .EQ. LGV)) MATCH = .TRUE.
                ENDDO
                IF(.NOT. MATCH) THEN
                  IC = IC+1
                  CTABLE(1,IC) = LMV
                  CTABLE(2,IC) = LGV
                ENDIF
              ENDIF
            ENDDO
          ENDIF
        ENDDO
      ENDDO
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - - -
      RETURN
      END
```

Subroutine rblade

```
      SUBROUTINE RBLADE(LENFN,BLADE_DB,LUGEOM,NTUBE,CBSPAN,ATID,ATOD
     2 ,CBTHICK,TRSPAN,TRTHICK,WSTHICK,CBLENGTH,NCS,CSOFF,CSWIDTH
     3 ,CBPMAT,ATMAT,CBSMAT,CBTRMAT)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * R B L A D E *  Reads Dataset Containing Control Blade
C - -                    Geometry Dataset
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        LENFN - Length of File Name                          (input)
C - -     BLADE_DB - Name of File Containing Appropriate Core-    (input)
C - -                layout Database
C - -       LUGEOM - Logical Unit Number for Blade Geometry        (input)
C - -                Dataset
C - -        NTUBE - Number of Absorber Tubes in All Four         (output)
C - -                Wings of Blade
C - -       CBSPAN - Control Blade Span (Tie Rod Center to        (output)
C - -                Tip)
C - -         ATID - Absorber Tube Inner Diameter (cm)             (output)
C - -         ATOD - Absorber Tube Outer Diameter (cm)             (output)
C - -      CBTHICK - Blade Wing Thickness (cm)                     (output)
C - -       TRSPAN - Tie Rod Span (cm)                             (output)
C - -      TRTHICK - Tie Rod Thickness (cm)                        (output)
C - -      WSTHICK - Wing Thickness (cm)                           (output)
C - -     CBLENGTH - Active Absrober Length (cm)                   (output)
C - -          NCS - Number of Central Stiffeners per Wing         (output)
C - -        CSOFF - Central Stiffener Offset                      (output)
C - -      CSWIDTH - Central Stiffener Width                       (output)
C - -       CBPMAT - Identifier for Blade Poison Material          (output)
C - -        ATMAT - Identifier for Blade Absorber Tube Mat-       (output)
C - -                erial
C - -       CBSMAT - Identifier for Blade Sheath Material          (output)
C - -      CBTRMAT - Identifier for Blade Tie Rod Material         (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
      CHARACTER*6 CBPMAT,ATMAT,CBSMAT,CBTRMAT
      CHARACTER*(LENFN) BLADE_DB
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /BLADE/ NTUBE,CBSPAN,ATID,ATOD,CBTHICK,TRSPAN,TRTHICK
     2 ,CBLENGTH,WSTHICK,NCS,CSOFF,CSWIDTH,CBPMAT,ATMAT,CBSMAT,CBTRMAT
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FTOPEN(LUGEOM,BLADE_DB)
C - - Skip Record Containing Dataset Title
      READ(LUGEOM,'(1X)')
C - - Read NAMELIST Input Block
      READ(LUGEOM,BLADE)
C - - Close Dataset
      CALL FTCLOSE(LUGEOM)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

Subroutine readin

```
      SUBROUTINE READIN(FIN,LENFN,CORE_DB,NIN,LUCGEOM,NAXIAL,AFL,NROW
     2 ,NCOL,NROWP,NCOLP,APITCH,SOD,STHICK,VOD,VTHICK,TUTPR,TCGR,BLTPR
```

```
     3 ,BCPR,LPREFIX,FPREFIX,CORE_MTLS,MVESSEL,MSHROUD,MTG,MCP,MUTP
     4 ,MLTP,CORE_F,BLADE_DB,MIGT,DTOD,DTID,RHO,RHOBYP,TEMPK)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * R E A D I N *  Reads NAMELIST input to program -- including
C - -                    the core geometry file (defined as function
C - -                    for compatibility with C invocation)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -          FIN - File Name for Input File                    (input)
C - -        LENFN - Length of File Name                         (input)
C - -      CORE_DB - Name of File Containing Appropriate Core-  (output)
C - -                layout Database
C - -          NIN - Logical Unit Number for Input File          (input)
C - -      LUCGEOM - Logical Unit Number for Core Geometry       (input)
C - -                Dataset File
C - -       NAXIAL - Number of Axial Nodes in Active Fuel       (output)
C - -          AFL - Active Fuel Length (cm)                    (output)
C - -         NROW - Number of Rows in Full Core                (output)
C - -         NCOL - Number of Columns in Full Core             (output)
C - -        NROWP - Number of Rows in Core Represented in the (output)
C - -                Problem
C - -        NCOLP - Number of Columnes in the Core Repre-      (output)
C - -                sented in the Problem
C - -       APITCH - Assembly Pitch (cm)                        (output)
C - -          SOD - Outer Radius of Core Shroud (cm)           (output)
C - -       STHICK - Core Shroud Thickness (cm)                 (output)
C - -          VOD - Pressure Vessel Outer Radius (cm)          (output)
C - -       VTHICK - Pressure Vessel Thickness (cm)             (output)
C - -        TUTPR - Top of the Upper Tie Plate Region (cm)     (output)
C - -          TCGR - Top of the Core Grid Region (cm)          (output)
C - -         BLTPR - Bottom of the Lower Tie Plate Region (cm) (output)
C - -          BCPR - Bottom of the Core Plate Region (cm)      (output)
C - -      LPREFIX - File Description Prefix for Lattice Geo-   (output)
C - -                metry Files
C - -      FPREFIX - File Description Prefix for Fuel Materi    (output)
C - -                al Intermediate Datasets
C - -    CORE_MTLS - File Specification for Core Materials      (output)
C - -                Dataset
C - -      MVESSEL - Material Identifier for Vessel             (output)
C - -      MSHROUD - Material Identifier for Core Shroud        (output)
C - -          MTG - Material Identifier for Core Top Guide     (output)
C - -                Region
C - -          MCP - Material Identifier for Core Plate         (output)
C - -                Region
C - -         MUTP - Material Identifier for Upper Tie Plate    (output)
C - -                Region
C - -         MLTP - Material Identifier for Lower Tie Plate    (output)
C - -                Region
C - -       CORE_F - Fraction of Core in Problem                (output)
C - -                (1 - full, 2 - half, 4 - quarter)
C - -     BLADE_DB - Dataset for Control Blade used in the      (output)
C - -                Problem
C - -         MIGT - Material Identifier for Incore Guide Tube (output)
C - -         DTOD - Incore Guide Tube Outer Diameter (cm)      (output)
C - -         DTID - Incore Guide Tube Inner Diameter (cm)      (output)
```

```
C - -          RHO - In-channel Water Density (g/cc)          (output)
C - -       RHOBYP - Bypass Water Density (g/cc)              (output)
C - -        TEMPK - Problem Temperature (Kelvin)             (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - -
C - -     NIN - Logical Unit Number for Input File
C - - LUCGEOM - Logical Unit Number for Core-geometry Dataset File
      INTEGER NIN,CORE_F
      CHARACTER*6 MVESSEL,MSHROUD,MTG,MCP,MUTP,MLTP,MIGT
      CHARACTER*132 CORE_DB,LPREFIX,FPREFIX,CORE_MTLS,BLADE_DB
      CHARACTER*(LENFN) FIN
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /LINKIN/ CORE_DB,NAXIAL,AFL,NROWP,NCOLP,LPREFIX,FPREFIX
     2 ,CORE_MTLS,CORE_F,BLADE_DB,RHO,RHOBYP,TEMPK,MUTP,MLTP
      NAMELIST /CORE/ NCOL,NROW,APITCH,SOD,STHICK,VOD,VTHICK,TUTPR
     2 ,TCGR,BLTPR,BCPR,MVESSEL,MSHROUD,DTOD,DTID,MTG,MCP,MIGT
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FTOPEN(NIN,FIN)
C - - Skip Record Containing Case Title
      READ(NIN,'(1X)')
C - - Read NAMELIST Input Block
      READ(NIN,LINKIN)
C - - Process Core-geometry File Namelist Input Section - - - - - - - -
      CALL FTOPEN(LUCGEOM,CORE_DB)
C - - Skip Header Record
      READ(LUCGEOM,'(1X)')
C - - Read NAMELIST Input
      READ(LUCGEOM,CORE)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

Subroutine rlattice

```
      SUBROUTINE RLATTICE(LENFN,FUEL_DB,LUGEOM,CTHICK,ASIN,WGAP,NGAP
     2 ,CRADIUS,FSRD,CFSRD,RPITCH,COD,CLD,POD,FRCMAT,FCMAT,LATDIM,NWR
     3 ,WROD,WRTH)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   * R L A T T I C E *  Reads Dataset Containing Lattice
C - -                        Geometry Dataset
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -   Argument(s):
C - -        LENFN - Length of File Name                        (input)
C - -      FUEL_DB - Name of File Containing Appropriate        (input)
C - -                Lattice-layout Database
C - -       CTHICK - Channel Thickness (cm)                    (output)
C - -         ASIN - Inner Span of Channel (cm)                (output)
C - -         WGAP - Wide Gap Thickness (cm)                   (output)
C - -         NGAP - Narrow Gap Thickness (cm)                 (output)
C - -      CRADIUS - Inner Radius of Channel Corner (cm)       (output)
C - -         FSRD - Clad Surface to Clad Surface Separation   (output)
C - -                (cm)
C - -        CFSRD - Clad Surface to Inner Channel Surface     (output)
C - -                Separation (cm)
C - -       RPITCH - Fuel Rod Pitch (cm)                       (output)
```

```
C - -          COD - Cladding Outer Diameter (cm)                (output)
C - -          CLD - Cladding Thickness (cm)                     (output)
C - -          POD - Fuel Pellet Outer Diameter (cm)             (output)
C - -       FRCMAT - Material Identifier for Fuel Rod Cladding   (output)
C - -        FCMAT - Material Identifier for Channel             (output)
C - -       LATDIM - Lattice Dimensionality                      (output)
C - -          NWR - Number of Water Rods                        (output)
C - -         WROD - Water Rod Outer Diameter (cm)               (output)
C - -         WRTH - Water Rod Thickness (cm)                    (output)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C - -
C - - Type Statement(s) - - - - - - - - - - - - - - - - - - - - - - - - -
      REAL NGAP
      CHARACTER*6 FRCMAT,FCMAT
      CHARACTER*(LENFN) FUEL_DB
C - - Initialization - - - - - - - - - - - - - - - - - - - - - - - - - -
      WROD = 0.0
      WRTH = 0.0
C - - Namelist Statements - - - - - - - - - - - - - - - - - - - - - - - -
      NAMELIST /FUEL/ CTHICK,ASIN,WGAP,NGAP,CRADIUS,FSRD,CFSRD,RPITCH
     2 ,COD,CLD,POD,LATDIM,NWR,FRCMAT,FCMAT,WROD,WRTH
C - - Read in Namelist - - - - - - - - - - - - - - - - - - - - - - - - -
      CALL FTOPEN(LUGEOM,FUEL_DB)
C - - Skip Record Containing Dataset Title
      READ(LUGEOM,'(1X)')
C - - Read NAMELIST Input Block
      READ(LUGEOM,FUEL)
C - - Close Dataset
      CALL FTCLOSE(LUGEOM)
C - - End of Normal Processing - - - - - - - - - - - - - - - - - - - - -
   10 RETURN
      END
```

2.4. Input Editing Routines

Function edit_ct

```
#include <stdio.h>
#include <string.h>

void edit_ct(int nlatticm,int *p){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  e d i t _ c t  *  Edits Contents of Correspondence Table, which
 - -                      Relates Lattice Geometry Indices to Lattice
 - -                      Material Indices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -   nlatticm - Number of Unique Lattice Material Indices      (input)
 - -          p - Pointer to Correspondence Table                (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  n - index for loop
 - - p2 - pointer to second member of correspondence table
*/
  int n, *p2;
/* - File Variable(s)
 - -  nout - output stream
*/
  extern FILE *nout;
/* - Edit Correspondence Table - - - - - - - - - - - - - - - - - - - - - - - */
  lines(7);
  fprintf(nout,"0Correspondence Table for Lattice Material and Geometry");
  fprintf(nout," Indices\n");
  fprintf(nout,"0Index Material Geometry\n\n");
  p2 = p+1;
  for( n = 1; n <= nlatticm; n++){
    if(n > 7) lines(1);
    fprintf(nout," %3i   %6i   %6i\n",n,*p,*p2);
    p++; p++;
    p2 = p+1;}
}
```

Function edit_spacer

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

void edit_spacer(int nbundlg,int *ptr_n_spacer
 ,float *ptr_spacer_location,ascii_string *ptr_spacer_material){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  e d i t _ s p a c e r  *  Edits Input Variables Defining
 - -                               Fuel Assembly Spacers
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -    nbundlg - number of fuel assembly geometrical types    (input)
```

```
 - - ptr_n_spacer
 - -            - number of spacers in each fuel assembly type (input)
 - - ptr_spacer_location
 - -            - pointer to array containing location of spa- (input)
 - -              cers for each fuel assembly type
 - - ptr_spacer_material
 - -            - pointer to array containing spacer material  (input)
 - -              mnemonic for each fue assembly type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -      i - utility index variable
 - -      j - utility index variable
 - -  ptr_n - pointer to integer vector
*/
  short int i,j;
  int *ptr_n = ptr_n_spacer;
/* - Float Variable(s)
 - - ptr_f - pointer to floating point array
*/
  float *ptr_f = ptr_spacer_location;
/* - Character Variable(s)
 - -  ptr_c - pointer to ascii_string vector
 - - buffer - character variable for processing output
*/
  ascii_string buffer, *ptr_c = ptr_spacer_material;
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Summarize NAMELIST Input Directives - - - - - - - - - - - - - - - - */
  header();
  lines(5+nbundlg);
  fprintf(nout
   ,"0Fuel Assembly Spacer Information\n");
/* - Number of Spacers for Each Unique Fuel Assembly Geometry */
  fprintf(nout,"0Index  Number of Spacers  Material Mnemonic\n\n");
  for(i = 1;i <= nbundlg;i++){
    strcpy(buffer,"");
    strncpy(buffer,*ptr_c,7);
    fprintf(nout," %3i          %2i                  %s\n"
     ,i,*ptr_n,buffer);
    ptr_n++;
    ptr_c++;}
/* - Spacer Coordinates for Each Type */
  lines(3+nbundlg);
  fprintf(nout,"0Index Axial Distance from BAF (cm)\n\n");
  ptr_n = ptr_n_spacer;
  for(i = 1;i <= nbundlg;i++){
    fprintf(nout," %3i",i);
    for(j = 1;j <= *ptr_n;j++){
      fprintf(nout," %7.3f",*ptr_f);
      *ptr_f++;
    }
```

```
    fprintf(nout,"\n");
    ptr_n++;
  }
}
```

Function fgds_edt

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct fuel_gometry{
                              ascii_string gds_name;
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              float wrod;
                              float wrth;
                              char frcmat[6];
                              char fcmat[6];
                            } fg_list;

  void fgds_edt(int nlatticg,char lprefix[],ascii_string *lgdsnam
   ,fg_list *ptr_lg_ds){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - fgds_edt - edits contents of fuel geometry datasets
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - - operation - flag for memory operation to perform            (input)
 - -   lprefix - prefix for lattice geometry datasets            (input)
 - -   lgdsnam - file descriptions for lattice geometry datasets (input)
 - - ptr_lg_ds - pointer to first element in list of fuel geo-   (input)
 - -             datasets
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variables
 - -   i - loop control variable
*/
  short int i;
/* - Character Variable(s)
 - -     fn - filename for lattice geometry dataset
 - - buffer - string used to manage output
*/
  ascii_string fn, buffer;
/* - FILE Variable(s)
```

```
 - -  nout - stream for output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  p - pointer to current element in list of fuel geometry datasets
 - -  l - pointer to current element in list of lattice gometry dataset
 - -      names
*/
  fg_list *p = ptr_lg_ds;
  ascii_string *l = lgdsnam;
/* - Loop Over all the Lattice Geometry Types - - - - - - - - - - - - - - */
  for( i = 1; i <= nlatticg; i++,p++,l++){
    if(p->wrod != 0)
      lines(20);
    else
      lines(22);
    strcpy(fn,lprefix);
    strcat(fn,*l);
    fprintf(nout,"0Values from Lattice Geometry Dataset:\n  %s\n",fn);
    fprintf(nout,"0 Variable     Value    Definition\n\n");
    sprintf(buffer,"    LATDIM    %6i",p->latdim);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Lattice Dimensionality\n");
    fprintf(nout,buffer);
    sprintf(buffer,"       NWR    %6i",p->nwr);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Number of Water Rods\n");
    fprintf(nout,buffer);
    sprintf(buffer,"    CTHICK     %5.2f",p->cthick);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Channel Thickness (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"      ASIN     %5.2f",p->asin);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Channel Interior Span (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"      WGAP     %5.2f",p->wgap);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Wide Gap Half-thickness (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"      NGAP     %5.2f",p->ngap);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Narrow Gap Half-thickness (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"   CRADIUS     %5.2f",p->cradius);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Channel Corner Inner Radius (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"      FSRD     %5.2f",p->fsrd);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer,"Clad Surface to Clad Surface Separation (cm)\n");
    fprintf(nout,buffer);
    sprintf(buffer,"     CFSRD     %5.2f",p->cfsrd);
    bufferpad(buffer,strlen(buffer),23);
    strcat(buffer
```

```
    ,"Clad Surface to Inner Channel Surface Separation (cm)\n");
   fprintf(nout,buffer);
   sprintf(buffer,"    RPITCH     %5.2f",p->rpitch);
   bufferpad(buffer,strlen(buffer),23);
   strcat(buffer,"Fuel Rod Pitch (cm)\n");
   fprintf(nout,buffer);
   sprintf(buffer,"       COD     %5.2f",p->cod);
   bufferpad(buffer,strlen(buffer),23);
   strcat(buffer,"Fuel Rod Cladding Outer Diameter (cm)\n");
   fprintf(nout,buffer);
   sprintf(buffer,"       CLD     %5.2f",p->cld);
   bufferpad(buffer,strlen(buffer),23);
   strcat(buffer,"Fuel Rod Cladding Thickness (cm)\n");
   fprintf(nout,buffer);
   sprintf(buffer,"       POD     %5.2f",p->pod);
   bufferpad(buffer,strlen(buffer),23);
   strcat(buffer,"Fuel Pellet Outer Diameter (cm)\n");
   fprintf(nout,buffer);
   sprintf(buffer,"    FRCMAT     %s",p->frcmat);
   bufferpad(buffer,strlen(buffer),23);
   strcat(buffer,"Fuel Rod Cladding Material Identifier\n");
   fprintf(nout,buffer);
   sprintf(buffer,"     FCMAT     %s",p->fcmat);
   bufferpad(buffer,strlen(buffer),23);
   strcat(buffer,"Fuel Channel Material Identifier\n");
   fprintf(nout,buffer);
   if(p->wrod != 0.0){
     sprintf(buffer,"      WROD     %5.2f",p->wrod);
     bufferpad(buffer,strlen(buffer),23);
     strcat(buffer,"Water Rod Outer Diameter (cm)\n");
     fprintf(nout,buffer);
     sprintf(buffer,"      WRTH     %5.2f",p->wrth);
     bufferpad(buffer,strlen(buffer),23);
     strcat(buffer,"Water Rod Thickness (cm)\n");
     fprintf(nout,buffer);}
  }
}
```

2.5. Deck Generation Routines

Function add_surface

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;

surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);

void add_surface(int operation,ascii_string label,char mnemonic[4]
 ,FILE *lu8,FILE *lu9,int *index,ascii_string value
 ,surface_usage_list *ptr_surface_usage,int *nsurface){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  a d d _ s u r f a c e  *  adds surface to MCNP model
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -  operation - operation to perform                          (input)
 - -              (0 - add new surface card,
 - -               1 - do not add new surface card)
 - -      label - Label Surface                                 (input)
 - -   mnemonic - MCNP mnemonic for surface (first edit only)   (input)
 - -        lu8 - Pointer to Scratch File Stream for Cell Def-  (input)
 - -              initions
 - -        lu9 - Pointer to Scratch File Stream for Surface    (input)
 - -              Definitions
 - -      index - Index for Surface                              (i|o)
 - -      value - String Containing Values for Defining Surface (input)
 - - ptr_surface_usage
 - -            - Points to Surface Usage Linked List           (input)
 - -   nsurface - total number of surfaces                       (i|o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        multiplier - multiplier on index for cell card
 - -  equivalent_index - index for equivalent surface
*/
  int multiplier = 1.0, equivalent_index;
/* - Character Variable(s)
 - -           buffer - buffer for processing lines of output
```

```
 - -          ptr_buf - pointer to buffer
 - - equivalent_label - label for surface with identical definition
*/
  ascii_string buffer, equivalent_label = "";
  char *ptr_buf;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Structured Variable(s) - - - - - - - - - - - - - - - - - - - - - - -
 - -  ptr_sl - pointer to surface usage list
*/
  surface_usage_list *ptr_sl;
/* - For New Surfaces,  Write to Surface Accumulation Scratch File */
  if(*index < 0) {
    multiplier = -1;
    *index = -(*index);}
  if(operation == 0){
/* - Search for Equivalent Surface that has already been Defined */
    search_surface_usage_list(2,"",&equivalent_index,value,mnemonic
     ,equivalent_label,ptr_surface_usage);
    ptr_sl = ptr_surface_usage;
    while(ptr_sl->next != NULL) ptr_sl = ptr_sl->next;
    if(equivalent_index == 0){
      (*nsurface)++;
      *index = *nsurface;}
    else
      *index = equivalent_index;
    ptr_sl = load_surface_usage_list(label,*index,value,mnemonic
     ,equivalent_label,ptr_sl);
    if(equivalent_index == 0){
      fprintf(lu9,"c     %s\n",label);
      fprintf(lu9,"%5i %s %s\n",*index,mnemonic,value);}
  }
/* - Write Surface Definition to Cell Card in MCNP Input Deck - - - -  */
  fprintf(lu8," %i",(multiplier*(*index)));
/* - Write to Edit */
  sprintf(buffer,"       %6i %s",(multiplier*(*index)),mnemonic);
  bufferpad(buffer,strlen(buffer),59);
  ptr_buf = buffer+59*sizeof(char);
  sprintf(ptr_buf,"%s\n",label);
  lines(1);
  fprintf(nout,buffer);
}
```

Function augment_lattice_list

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];

typedef struct all{
                    struct all *last;
                    int basis_lattice_material_index;
                    int lattice_material_index;
```

```
                    struct all *next;
                   } augmented_lattice_list;


augmented_lattice_list *memory_lattice_list(int
 ,augmented_lattice_list *);

void augment_lattice_list(int *nlatticm,int *lmvect,int *lgvect
 ,int nlatticm_ref,int nlatticg,int nbundlm,int nbundlg,int naxial
 ,int *ptr_n_spacer,int *ptr_spacer_node
 ,augmented_lattice_list *additional_lattices
 ,int *ptr_correspondence_table){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - * a u g m e n t _ l a t t i c e _ l i s t *  adds lattices incor-
 - -                                               porating spacer grids
 - -                                               into material lattice
 - -                                               loading vectors
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -   nlatticm - total number of unique material lattice indices  (i&o)
 - -     lmvect - material lattice loading vectors                 (i&o)
 - -     lgvect - geometrical lattice loading vectors             (input)
 - -  nlatticm_ref
 - -            - total number of unique material lattice indi-  (input)
 - -              ces without augmentation for lattices incor-
 - -              porating spacers
 - -   nlatticg - total number of unique geometrical lattice in- (input)
 - -              dices
 - -    nbundlm - number of unique fuel assembly materials        (input)
 - -    nbundlg - number of unique fuel assembly geometries       (input)
 - -     naxial - number of axial nodes in problem                (input)
 - - ptr_n_spacer
 - -            - number of spacer grids associated with each    (input)
 - -              geometrically unique fuel assembly
 - - ptr_spacer_node
 - -            - nodal location of spacer grids for each geo-   (input)
 - -              metric fuel assembly type
 - - additional_lattices
 - -            - linked list of additional lattices generated  (output)
 - -              to accomodate spacer grids
 - - ptr_correspondence_table
 - -            - table providing relationship between lattice   (input)
 - -              material indices and lattice geometrical in-
 - -              dices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -              k - axial index for looping
 - -              n - uility loop variable
 - -          nglat - lattice geometry index
 - -          ngasm - fuel assembly geometry index
 - -            lmv - pointer to lattice material index list
 - -            lgv - pointer to lattice geometry index list
 - - spacer_present - flag to indicate whether a spacer grid is present
```

```
 - -                    at this location
*/
  short int k,n;
  int ngasm, spacer_present, nglat;
  int *lmv = lmvect, *lgv;
/* - Structured Variable(s)
 - -  ptr_al - pointer to additional lattice linked list
*/
  augmented_lattice_list *ptr_al = additional_lattices;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - pointer to output stream
*/
  extern FILE *nout;
/* - Sweep through Material Assembly Types - - - - - - - - - - - - - - - - */
  for(n = 1;n <= nbundlm;n++){
/* - Skip Fuel Assembly Index */
    lmv++;
/* - Sweep from Bottom of Assmbly to Top */
    for(k = 1;k <= naxial;k++,lmv++){
/* - Determine Corresponding Lattice Geometrical Index */
      { short int i;
        int *p = ptr_correspondence_table;
        while(*p != *lmv) {p++;p++;}
        nglat = *(p+1);
      }
/* - Find Fuel Assembly Geometrical Type of which the Lattice Geometry
 - - is a Member */
      lgv = lgvect;
      { short int i,bk,ns;
        int *ptr = ptr_n_spacer;
        int *ptr_n = ptr_spacer_node;
        ngasm = 0;
        for(i = 1;i <= nbundlg;i++)
          if(ngasm == 0){
/* - Skip Fuel Assembly Index */
            lgv++;
            for(bk = 1;bk <= naxial;bk++,lgv++)
              if(*lgv == nglat){
                ngasm = i;
                break;}
          }
          else break;
/* - Compare Nodes at which Spacer Grids are Located with Current Axial
 - - Node Index */
        spacer_present = 0;
        for(i = 1;i < ngasm;i++,ptr++)
          for(ns = 1;ns <= *ptr;ns++,ptr_n++);
        for(bk = 1;bk <= *ptr;bk++,ptr_n++)
          if(k == *ptr_n){
            spacer_present = 1;
            break;}
      }
/* - Determine whether this Lattice Incorporating a Spacer Grid has
 - - already been defined */
      if(spacer_present == 1){
```

```
        { short int found = 0, first = 0;
          augmented_lattice_list *ptr_next;
          ptr_al = additional_lattices;
          do{
              if(first != 0) ptr_al = ptr_al->next;
              first = 1;
              if(ptr_al->basis_lattice_material_index == *lmv){
                found = 1;
                *lmv = ptr_al->lattice_material_index;
                break;}
            }
          while(ptr_al->next != NULL);
/* - Spacer Grid not already Defined, Augment Lattice List */
          if(found == 0){
            ptr_al = additional_lattices;
/* - Check for First Entry in Linked List */
            if(ptr_al->lattice_material_index != NULL){
              while(ptr_al->next != NULL) ptr_al = ptr_al->next;
              ptr_next = memory_lattice_list(1,ptr_next);
              ptr_next->last = ptr_al;
              ptr_al->next = ptr_next;
              ptr_al = ptr_next;}
            (*nlatticm)++;
            ptr_al->basis_lattice_material_index = *lmv;
            ptr_al->lattice_material_index = *nlatticm;
            ptr_al->next = NULL;
            *lmv = *nlatticm;
          }
        }
      }
    }
  }
}
```

Function build_assemblies

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;

void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void lines(int);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
```

```
void build_assemblies(int *ncell,int *nuniverse,FILE *lu8,FILE *lu9
 ,surface_usage_list *ptr_surface_usage,int **ptr_ufa
 ,int nbundlm,int *lmvect,int naxial,int *ptr_ufl,int *nsurface
 ,int *ptr_n_node,float *ptr_node_boundaries,int *lgvect
 ,int *ptr_correspondence_table,int nlatticm,int nbundlg){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  b u i l d _ a s s e m b l i e s  *  builds fuel assemblies from
 - -                                        constituent lattices
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -       ncell - number of cells in MCNP model                    (input)
 - -   nuniverse - number of "universes" in MCNP model              (input)
 - -         lu8 - stream pointer for scratch file for cell        (input)
 - -               definition accumulation
 - -         lu9 - stream pointer for scratch file for surface     (input)
 - -               definition accumulation
 - - ptr_surface_usage
 - -             - linked list for surfaces already defined in     (input)
 - -               the problem
 - -     ptr_ufa - vector with universe indices for unique fuel (output)
 - -               assembly types
 - -     nbundlm - number of unique fuel assembly types            (input)
 - -      lmvect - vector containing assignment of lattice mat-    (input)
 - -               erial indices to fuel assembly material indi-
 - -               ces
 - -      naxial - number of axial nodes                           (input)
 - -     ptr_ufl - vector with universe indices for unique lat-    (input)
 - -               tice types
 - -    nsurface - total number of surfaces                        (input)
 - -  ptr_n_node - number of axial nodes for each geometrical      (input)
 - -               fuel assembly type
 - - ptr_node_boundaries
 - -             - locations of tops of axial nodes for each       (input)
 - -               geometrical fuel assembly type
 - -      lgvect - vector containing assignment of lattice geo-    (input)
 - -               metrical indices to fuel assembly material
 - -               indices
 - - ptr_correspondence_table
 - -             - tables relating lattice material indices to     (input)
 - -               lattice geometrical indices
 - -    nlatticm - number of unique material lattice types         (input)
 - -     nbundlg - number of unique geometrical fuel assembly      (input)
 - -               types
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -          i - utility loop variable
 - -          n - loop variable
 - -          k - loop control for axial location
 - -      index - utility index variable
 - -      ngasm - index for geometical assembly type
 - -        lmv - pointer to lmvect vector
 - -    ptr_lmv - second pointer to lmvect vector
```

```
 - -  ptr_table - pointer to correspondence table
 - -      ptr_n - pointer to vector containing number of axial nodes
 - -              for each geometrically unique fuel assembly
 - -          p - pointer to **ptr_ufa vector
*/
  short int i,n,k;
  int index, ngasm, *lmv = lmvect, *ptr_lmv = lmvect;
  int *ptr_lgv = lgvect;
  int *ptr_n = ptr_n_node;
  int *ptr_table = ptr_correspondence_table;
  int *p = *ptr_ufa;
/* - Float Variable(s)
 - -     ptr_nb - pointer to array containing the axial locations for
 - -              tops of axial nodes for each geometrically unique
 - -              fuel assembly
*/
  float *ptr_nb = ptr_node_boundaries;
/* - Character Variable(s)
 - -    label - label for cell, surface or material
 - - mnemonic - mnemonic for surfaces
 - - material - name of material
 - -    value - string for defining surface definition
*/
  char mnemonic[4];
  ascii_string label, material, value;
/* - FILE Variable(s)
 - -  nout - pointer to code output file
*/
  extern FILE *nout;
/* - Loop Over Unqiue Fuel Assemblies in the Core - - - - - - - - - -  */
  for(n = 1;n <= nbundlm;n++){
    (*nuniverse)++;
/* - Determine the Fuel Assembly Geometrical Index Corresonding to this
 - - Fuel Assembly Material Index; Start by Locating the Bottom Node
 Material Lattice Index in Correspondence Table */
    for(i = 1;i < n;i++)
      for(k = 1;k <= naxial;k++) ptr_lmv++;
    for(i = 1;i <= nlatticm;i++)
      if(*ptr_table == *ptr_lmv) break;
      else {ptr_table++; ptr_table++;}
/* - Determine the Fuel Assembly Geometrical Type of which the Lattice
 - - Geometrical Type is a Member */
    ngasm = 0;
    ptr_lgv = lgvect;
    for(i = 1;i <= nbundlg;i++){
      for(k = 1;k <= naxial;k++)
        if(*ptr_lgv == *ptr_table){
          ngasm = i;
          break;}
        else
          ptr_lgv++;
      if(ngasm != 0) break;
    }
    ptr_n = ptr_n_node;
    ptr_nb = ptr_node_boundaries;
```

```
    for(i = 1;i < ngasm;i++){
      for(k = 1;k <= *ptr_n;k++) ptr_nb++;
      ptr_n++;}
/* - Loop Over all the Nodes in an Assembly */
    for(k = 1;k <= naxial;k++){
/* - Cells for Axial Nodes */
      sprintf(label,"Axial Node %i for Fuel Assembly %i",k,n);
      strcpy(material,"void");
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
/* - Surfaces for Node Boundaries */
      if(k != 1) {
        sprintf(label,"Top of Node %i",(k-1));
        search_surface_usage_list(1,label,&index,"","","""
         ,ptr_surface_usage);
        if(index == 0){
          lines(3);
          fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
          fprintf(nout," populate_control_cells --\n");
          fprintf(nout
           ," Surface not Found in Linked List, label = %s\n"
           ,label);}
        add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
         ,nsurface);}
      if(k != naxial){
        sprintf(label,"Top of Node %i",k);
        search_surface_usage_list(1,label,&index,"","","""
         ,ptr_surface_usage);
        if(index == 0){
          strcpy(mnemonic,"pz");
          sprintf(value,"%10.4E",*ptr_nb);
          index = -1;
          add_surface(0,label,mnemonic,lu8,lu9,&index,value
           ,ptr_surface_usage,nsurface);}
        else{
          index = -index;
          add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
           ,nsurface);}
        ptr_nb++;
      }
      fprintf(lu8,"\n");
      { short int nn;
        int *l = lmv;
        int *p = ptr_ufl;
        l++;
        for(nn = 1;nn < k;nn++,l++) continue;
        for(nn = 1;nn < *l;nn++,p++) continue;
        fprintf(lu8,"      fill= %i\n",*p);
        fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
      }
    }
/*    **ptr_ufa = *nuniverse; */
    *p = *nuniverse;
    p++;
    for(k = 1;k <= (naxial+1);k++) lmv++;
  }
```

```
}
```

Function ge7x7_lattice

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                            struct ascii_record *last;
                            ascii_string line;
                            struct ascii_record *next;
                           } a_record;
typedef struct s_material{
                          struct s_material *last;
                          int atomic_number;
                          int mass_number;
                          float weight_percentage;
                          char library_suffix[5];
                          struct s_material *next;
                         } ll_material;

typedef struct u_list{
                      struct u_list *last;
                      int index;
                      ascii_string label;
                      struct u_list *next;
                     } usage_list;

typedef struct su_list{
                       struct su_list *last;
                       int index;
                       ascii_string label;
                       ascii_string value;
                       char mnemonic[4];
                       ascii_string equivalent_label;
                       struct su_list *next;
                      } surface_usage_list;

void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
ll_material *material_match(a_record *,char[],float *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void abort();
void lines(int);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_like_but(char[],char[],FILE *,int *,char[],int,int *);

void ge7x7_lattice(float cthick,float asin,float wgap,float ngap
 ,float cradius,float fsrd,float cfsrd,float rpitch,float cod
 ,float cld,float pod,char frcmat[],char fcmat[],int latdim,int nft
```

```
,int *lattice,int *ncell,int *nmaterial,FILE *lu8,FILE *lu9
,FILE *lu10,surface_usage_list *ptr_surface_usage
,usage_list *ptr_material_usage,float inchannel_density
,float bypass_density,int *nuniverse,float *fp_density
,ascii_string matdsnam,ll_material *fmids,a_record *ptr_core_mtls
,int nlat,int *ufl,int *nsurface,char inchannel_material []){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - *  g e 7 x 7 _ l a t t i c e  *  Create Lattice Model for GE 7x7
- -                                  Lattice Model
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - Argument(s):
- -      cthick - Channel Thickness (cm)                              (input)
- -        asin - Inner Span of Channel (cm)                          (input)
- -        wgap - Wide Gap Thickness (cm)                             (input)
- -        ngap - Narrow Gap Thickness (cm)                           (input)
- -     cradius - Inner Radius of Channel Corner (cm)                 (input)
- -        fsrd - Clad Surface to Clad Surface Separation             (input)
- -               (cm)
- -       cfsrd - Clad Surface to Inner Channel Surface               (input)
- -               Separation (cm)
- -      rpitch - Fuel Rod Pitch (cm)                                 (input)
- -         cod - Cladding Outer Diameter (cm)                        (input)
- -         cld - Cladding Thickness (cm)                             (input)
- -         pod - Fuel Pellet Outer Diameter (cm)                     (input)
- -      frcmat - Material Identifier for Fuel Rod Cladding           (input)
- -       fcmat - Material Identifier for Channel                     (input)
- -      latdim - Lattice Dimensionality                              (input)
- -         nft - Number of Unique Fuel Rod Types                     (input)
- -     lattice - Fuel Rod Type Map                                   (input)
- -       ncell - number of MCNP cells                                  (i&o)
- -   nmaterial - number of MCNP materials                              (i&o)
- -         lu8 - stream pointer to scratch file for cell defin-      (input)
- -               itions
- -         lu9 - stream pointer to scratch file for surface          (input)
- -               definitions
- -        lu10 - stream pointer to scratch file for material         (input)
- -               definitions
- - ptr_surface_usage
- -             - list of defined surfaces                            (input)
- - ptr_material_usage
- -             - list of defined materials                           (input)
- - inchannel_density
- -             - density for in-channel moderator (g/cc)             (input)
- - bypass_density
- -             - density for bypass moderator (g/cc)                 (input)
- -   nuniverse - number of MCNP "universes"                            (i&o)
- - fp_density - density for fuel pellets                             (input)
- -    matdsnam - name for fuel intermediate material dataset         (input)
- -       fmids - linked list for fuel material composition            (input)
- - ptr_core_mtls
- -             - pointer to linked list containing core mat-        (input)
- -               erial definitions
- -        nlat - index for unique lattice for which to create        (input)
- -               MCNP lattice model
- -         ufl - index for universe that contains the entire         (input)
```

```
 - -                lattice representation
 - -   nsurface - total number of surfaces                        (input)
 - - inchannel_material
 - -            - mnemonic for inchannel moderator material      (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         n - index for loops
 - -       len - length of ascii strings
 - -     index - utility variable for indices
 - - n_entries - number of entries in linked list for fuel material
 - -       ufr - universe index for first fuel rod
 - -      ufrl - universe index for fuel rod window lattice
 - -     uchan - universe index for channel and contents
 - -     cchan - cell index for channel
 - -      cwic - cell index for area inside channel
 - -  ncpellet - cell index for reference fuel pellet
 - -     ncgap - cell index for reference fuel/cladding gap
 - -    ncclad - cell index for reference cladding
 - -   ncwater - cell index for water outside of reference fuel rod
*/
  short int n;
  int len = 132, length, index;
  int n_entries;
  int ufr, ufrl, uchan;
  int cchan, cwic;
  int ncpellet, ncgap, ncclad, ncwater;
/* - Float Variable(s)
 - -    spor - reference fuel pellet outer surface
 - -    scir - reference cladding inner surface
 - -    scor - reference cladding outer surface
 - - xminfrw - XMIN surface for fuel rod window
 - - xmaxfrw - XMAX surface for fuel rod window
 - - yminfrw - XMIN surface for fuel rod window
 - - ymaxfrw - YMAX surfacr for fuel rod window
 - -  wgcowx - wide gap, channel outside wall, px surface
 - -  wgciwx - wide gap, channel inside wall, px surface
 - -  ngciwx - narrow gap, channel inside wall, px surface
 - -  ngcowx - narrow gap, channel outside wall, px surface
 - -  wgcowy - wide gap, channel outside wall, py surface
 - -  wgciwy - wide gap, channel inside wall, py surface
 - -  ngciwy - narrow gap, channel inside wall, py surface
 - -  ngcowy - narrow gap, channel outside wall, py surface
 - - xambig1 - ambiguity surface for channel corners (wide gap)
 - - xambig2 - ambiguity surface for channel corners (narrow gap)
 - - yambig1 - ambiguity surface for channel corners (wide gap)
 - - yambig2 - ambiguity surface for channel corners (narrow gap)
 - -    ccro - outer radius for channel corner
 - -    ccri - innner radius for channel corner
 - -    cc1x - center for channel corner 1
 - -    cc1y - center for channel corner 1
 - -    cc2x - center for channel corner 2
 - -    cc2y - center for channel corner 2
 - -    cc3x - center for channel corner 3
```

```
 - -    cc3y - center for channel corner 3
 - -    cc4x - center for channel corner 4
 - -    cc4y - center for channel corner 4
 - - density - density for material
 - - x_ofset - x-offset for fuel rod centering to base lattice position
 - - y_ofset - y-offset for fuel rod centering to base lattice position
*/
  float spor, scir, scor,xminfrw,xmaxfrw,yminfrw,ymaxfrw,wgcowx,wgciwx
   ,ngciwx,ngcowx,wgcowy,wgciwy,ngciwy,ngcowy,xambig1,xambig2,yambig1
   ,yambig2,ccro,ccri,cc1x,cc1y,cc2x,cc2y,cc3x,cc3y,cc4x,cc4y;
  float density,x_ofset,y_ofset;
/* - Character Variable(s)
 - -    label - label for cell or surface
 - - material - label for cell material for output edit
 - - mnemonic - MCNP mnemonic for surface definition
 - -    value - surface defintion values in string form
 - -   mdsnam - working pointer for lattice material intermediate
 - -           dataset name
*/
  char mnemonic[4];
  ascii_string label,material,value;
/* - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition linked list
*/
  ll_material *ptr_mtl;
/* - Compute Surfaces for Lattice - - - - - - - - - - - - - - - - - - - - */
  spor = pod/2;
  scir = (cod/2)-cld;
  scor = cod/2;
  xminfrw = -rpitch/2;
  xmaxfrw = rpitch/2;
  yminfrw = -rpitch/2;
  ymaxfrw = rpitch/2;
  wgcowx = wgap;
  wgciwx = wgap+cthick;
  ngciwx = wgap+cthick+asin;
  ngcowx = wgap+(2*cthick)+asin;
  wgcowy = -wgap;
  wgciwy = -(wgap+cthick);
  ngciwy = -(wgap+cthick+asin);
  ngcowy = -(wgap+(2*cthick)+asin);
  xambig1 = wgap+cthick+cradius;
  xambig2 = wgap+cthick+asin-cradius;
  yambig1 = -(wgap+cthick+cradius);
  yambig2 = -(wgap+cthick+asin-cradius);
  ccro = cradius+cthick;
  ccri = cradius;
  cc1x = wgap+cthick+cradius;
  cc1y = -(wgap+cthick+cradius);
  cc2x = wgap+cthick+asin-cradius;
  cc2y = -(wgap+cthick+cradius);
```

```
  cc3x = wgap+cthick+asin-cradius;
  cc3y = -(wgap+cthick+asin-cradius);
  cc4x = wgap+cthick+cradius;
  cc4y = -(wgap+cthick+asin-cradius);
/* - compute offset to move fuel rod to reference location */
  x_ofset = wgap+cthick+cfsrd+(cod/2);
  y_ofset = -x_ofset;
/* - Build Lattice Model Starting with Fuel Pellet - - - - - - - - - - */
  for( n = 0; n < nft; n++){
    (*nuniverse)++;
    if(n == 0) ufr = *nuniverse;
/* - Pellet Cell */
    sprintf(label,"Fuel Pellet, #%i, L%i",(n+1),nlat);
    sprintf(material,"%s (%i)",matdsnam,(n+1));
    search_usage_list(1,material,&index,ptr_material_usage);
    if(n == 0){
      if(index == 0){
        (*nmaterial)++;
        add_cell(label,material,lu8,ncell,*nmaterial,-(*fp_density)
         ,nuniverse);
        n_entries = 1;}
      else{
        add_cell(label,material,lu8,ncell,index,-(*fp_density)
         ,nuniverse);
        n_entries = 0;}
      ncpellet = *ncell;}
    else{
      if(index == 0){
        (*nmaterial)++;
        sprintf(value,"mat= %i rho= %10.4E"
         ,*nmaterial,-(*fp_density));
        n_entries = 1;}
      else{
        sprintf(value,"mat= %i rho= %10.4E"
         ,index,-(*fp_density));
        n_entries = 0;}
      add_like_but(label,material,lu8,ncell,value,ncpellet
       ,nuniverse);}
    if(index == 0){
      { usage_list *ptr_ml = ptr_material_usage;
        while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
        index = (ptr_ml->index)+1;
        ptr_ml = load_usage_list(material,index,ptr_ml);
      }
    }
    fp_density++;
/* - Pellet Surface */
    if(n == 0){
      strcpy(label,"");
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i"
       ,(n+1),nlat);
      strcpy(mnemonic,"c/z");
      strcpy(value,"");
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,spor);
      index = -1;
```

```
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Pellet Materials */
    sprintf(label,"%s (%i)",matdsnam,(n+1));
/* - Determine number of entries in Linked List */
    if(n_entries == 1){
      { ll_material *f = fmids;
        do{
          f = f->next;
          n_entries++;}
        while(f->next != NULL);
      }
      add_material(lu10,nmaterial,label,n_entries,fmids);
    }
    if(n < (nft+1)) fmids++;
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Fuel/Cladding Gap - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Fuel/Cladding Gap, #%i, L%i",(n+1),nlat);
    strcpy(material,"void");
    if(n == 0){
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
      ncgap = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncgap,nuniverse);
/* - Surfaces */
    if(n == 0){
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scir);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material -- void used */
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Cladding - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Cladding, #%i, L%i",(n+1),nlat);
    strcpy(material,frcmat);
    if(n == 0){
      search_usage_list(1,frcmat,&index,ptr_material_usage);
```

```
      n_entries = 0;
      if(index == 0){
        { usage_list *ptr_ml;
          ptr_mtl =
           material_match(ptr_core_mtls,frcmat,&density,&n_entries);
          (*nmaterial)++;
          ptr_ml = ptr_material_usage;
          while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
          index = (ptr_ml->index)+1;
          ptr_ml = load_usage_list(frcmat,index,ptr_ml);
        }
      }
      else{
        (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
        n_entries = 0;}
      add_cell(label,material,lu8,ncell,index,-density,nuniverse);
      ncclad = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncclad,nuniverse);
/* - Surface(s) */
    if(n == 0){
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scor);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
     }
/* - Material */
    if(n == 0){
      if(n_entries != 0){
        add_material(lu10,nmaterial,frcmat,n_entries,ptr_mtl);
        rollup_llm(ptr_mtl);
        n_entries = 0;}
    }
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Water Outside of Outer Surface of Cladding - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Water Outside Cladding #%i, L%i",(n+1),nlat);
    strcpy(material,inchannel_material);
    if(n == 0){
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
```

```
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Material Not Found in Linked List, material = %s\n",label);
        abort();}
      add_cell(label,material,lu8,ncell,index,-inchannel_density
       ,nuniverse);
      ncwater = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncwater,nuniverse);
/* - Surface */
    if(n == 0){
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
/* - Materials */
/* - In-channel Material should already exist as a material */
    if(n == 0) fprintf(lu8,"\n");
    fprintf(lu8,"        u= %i imp:n= 1.0\n",(ufr+n));
  }
/* - Create Lattice for Fuel Assembly and Populate with Previously
 - - Defined Fuel Rod Universes - - - - - - - - - - - - - - - - - -   */
/* - Cell */
  sprintf(label,"Reference Fuel Rod Cell, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  (*nuniverse)++;
  ufrl = *nuniverse;
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"XMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",(xmaxfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
```

```
  sprintf(label,"XMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",(xminfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",(yminfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",(ymaxfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8," lat= 1 u = %i imp:n= 1.0\n",*nuniverse);
  fprintf(lu8,"      fill= -1:%i -1:%i 0:0\n"
   ,latdim,latdim);
  lines(latdim+1);
  fprintf(nout,"     Universes Filling the Lattice:\n");
  { short int i,j;
    int *plattice = lattice;
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
    for(i = 1;i <= latdim;i++){
      fprintf(lu8,"        %3i ",*nuniverse);
      fprintf(nout,"      ");
      for(j = 1;j <= latdim;j++,plattice++){
        fprintf(lu8,"%3i ",(*plattice+ufr-1));
        fprintf(nout,"%3i ",(*plattice+ufr-1));}
      fprintf(lu8,"%3i\n",*nuniverse);
      fprintf(nout,"\n");}
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
  }
/* - Create Channel Model and Populate with Lattice - - - - - - - -  */
/* - Channel - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*nuniverse)++;
  uchan = *nuniverse;
  sprintf(label,"Channel, L%i",nlat);
  search_usage_list(1,fcmat,&index,ptr_material_usage);
  n_entries = 0;
  if(index == 0){
    { usage_list *ptr_ml;
      ptr_mtl =
       material_match(ptr_core_mtls,fcmat,&density,&n_entries);
      (*nmaterial)++;
      ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
```

```
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(fcmat,index,ptr_ml);
    }
  }
  else{
    (void) material_match(ptr_core_mtls,fcmat,&density,&n_entries);
    n_entries = 0;
  }
  add_cell(label,fcmat,lu8,ncell,index,-density,nuniverse);
  cchan = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",wgcowx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",wgciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",yambig1);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",yambig2);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,")":(");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",wgcowy);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",wgciwy);
```

```
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = 1;
sprintf(value,"%10.4E",xambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",xambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,")\n");
fprintf(lu8,"      (");
sprintf(label,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",ngciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",ngcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngcowx);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
```

```
/* - Channel Corners */
/* - Northwest Corner */
  sprintf(label,"Channel Corner Outer Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")(");
/* - Northeast Corner */
  sprintf(label,"Channel Corner Outer Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NE), L%i"
```

```
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Southeast Corner */
  sprintf(label,"Channel Corner Outer Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
```

```
    ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Southwest Corner */
  sprintf(label,"Channel Corner Outer Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",uchan);
/* - Materials */
  if(n_entries != 0){
    add_material(lu10,nmaterial,fcmat,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
    n_entries = 0;}
/* - Area Inside the Channel - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Active Fuel Area, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density,nuniverse);
  cwic = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
```

```
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8,"):(");
 sprintf(label
  ,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
```

```
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8," ):\n      (");
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")(");
  sprintf(label
   ,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","
```

```
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label
 ,"Channel Corner Inner Radius (NE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
```

```
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,");(");
sprintf(label
 ,"Channel Corner Inner Radius (SE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
  sprintf(label
   ,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
```

```
  search_surface_usage_list(1,label,&index,"","",""
   ;ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      fill= %i u= %i imp:n= 1.0\n",ufrl,uchan);
  lines(1);
  fprintf(nout,"     Filled with Universe %i\n",ufrl);
/* - Area Outside the Channel - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Water Outside of Channel, L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Regions Outside of Four Channel Walls */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NW Corner */
  fprintf(lu8,"       (");
  sprintf(label,"Channel Corner Outer Radius (NW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
```

```
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SE Corner */
  fprintf(lu8,"       (");
  sprintf(label,"Channel Corner Outer Radius (SE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
```

```
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
/* - Assigned Universe Number to Return Variable */
  *ufl = uchan;
}
```

Function ge8x8_lattice_lcwr

```
#include <stdio.h>
#include <string.h>
```

```
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                         struct su_list *last;
                         int index;
                         ascii_string label;
                         ascii_string value;
                         char mnemonic[4];
                         ascii_string equivalent_label;
                         struct su_list *next;
                       } surface_usage_list;

typedef struct fuel_geometry{
                              int latdim;
                              int nwr;
                              float cthick;
                              float asin;
                              float wgap;
                              float ngap;
                              float cradius;
                              float fsrd;
                              float cfsrd;
                              float rpitch;
                              float cod;
                              float cld;
                              float pod;
                              float wrod;
                              float wrth;
                              char frcmat[6];
                              char fcmat[6];
                             } fg_list;


void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
```

```
 ,surface_usage_list *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
ll_material *material_match(a_record *,char[],float *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void abort();
void lines(int);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_like_but(char[],char[],FILE *,int *,char[],int,int *);

void ge8x8_lattice_lcwr(float cthick,float asin,float wgap,float ngap
 ,float cradius,float fsrd,float cfsrd,float rpitch,float cod
 ,float cld,float pod,char frcmat[],char fcmat[],int latdim,int nft
 ,int *lattice,int *ncell,int *nmaterial,FILE *lu8,FILE *lu9
 ,FILE *lu10,surface_usage_list *ptr_surface_usage
 ,usage_list *ptr_material_usage,float inchannel_density
 ,float bypass_density,int *nuniverse,float *fp_density
 ,ascii_string matdsnam,ll_material *fmids,a_record *ptr_core_mtls
 ,int nlat,int *ufl,int *nsurface,char inchannel_material []
 ,float wrod,float wrth,int *ncell_tr){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e 8 x 8 _ l a t t i c e _ l c w r  *  Create Lattice Model for
 - -                                            GE 8x8 Lattice with a
 - -                                            Large Central Water
 - -                                            Rod
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      cthick - Channel Thickness (cm)                          (input)
 - -        asin - Inner Span of Channel (cm)                      (input)
 - -        wgap - Wide Gap Thickness (cm)                         (input)
 - -        ngap - Narrow Gap Thickness (cm)                       (input)
 - -     cradius - Inner Radius of Channel Corner (cm)             (input)
 - -        fsrd - Clad Surface to Clad Surface Separation         (input)
 - -               (cm)
 - -       cfsrd - Clad Surface to Inner Channel Surface           (input)
 - -               Separation (cm)
 - -      rpitch - Fuel Rod Pitch (cm)                             (input)
 - -         cod - Cladding Outer Diameter (cm)                    (input)
 - -         cld - Cladding Thickness (cm)                         (input)
 - -         pod - Fuel Pellet Outer Diameter (cm)                 (input)
 - -      frcmat - Material Identifier for Fuel Rod Cladding       (input)
 - -       fcmat - Material Identifier for Channel                 (input)
 - -      latdim - Lattice Dimensionality                          (input)
 - -         nft - Number of Unique Fuel Rod Types                 (input)
 - -     lattice - Fuel Rod Type Map                               (input)
 - -       ncell - number of MCNP cells                              (i&o)
 - -   nmaterial - number of MCNP materials                          (i&o)
 - -         lu8 - stream pointer to scratch file for cell defin-  (input)
 - -               itions
 - -         lu9 - stream pointer to scratch file for surface      (input)
 - -               definitions
 - -        lu10 - stream pointer to scratch file for material     (input)
```

```
 - -                 definitions
 - - ptr_surface_usage
 - -               - list of defined surfaces                        (input)
 - - ptr_material_usage
 - -               - list of defined materials                       (input)
 - - inchannel_density
 - -               - density for in-channel moderator (g/cc)         (input)
 - - bypass_density
 - -               - density for bypass moderator (g/cc)             (input)
 - -  nuniverse - number of MCNP "universes"                           (i&o)
 - - fp_density - density for fuel pellets                            (input)
 - -   matdsnam - name for fuel intermediate material dataset        (input)
 - -       fmids - linked list for fuel material composition         (input)
 - - ptr_core_mtls
 - -               - pointer to linked list containing core mat-     (input)
 - -                 erial definitions
 - -        nlat - index for unique lattice for which to create      (input)
 - -                 MCNP lattice model
 - -         ufl - index for universe that contains the entire       (input)
 - -                 lattice representation
 - -   nsurface - total number of surfaces                            (input)
 - - inchannel_material
 - -               - mnemonic for inchannel moderator material       (input)
 - -        wrod - outer diameter for water rod (cm)                 (input)
 - -        wrth - water rod thickness (cm)                          (input)
 - -   ncell_tr - number of MCNP cells for translated cells            (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -          n - index for loops
 - -        len - length of ascii strings
 - -      index - utility variable for indices
 - - n_entries - number of entries in linked list for fuel material
 - -        ufr - universe index for first fuel rod
 - -       ufrl - universe index for fuel rod window lattice
 - -      uchan - universe index for channel and contents
 - -      cchan - cell index for channel
 - -       cwic - cell index for area inside channel
 - -   ncpellet - cell index for reference fuel pellet
 - -      ncgap - cell index for reference fuel/cladding gap
 - -     ncclad - cell index for reference cladding
 - -    ncwater - cell index for water outside of reference fuel rod
 - -      uwrnw - universe index for northwest quadrant of water rod
 - -      uwrne - universe index for northeast quadrant of water rod
 - -      uwrse - universe index for southeast quadrant of water rod
 - -      uwrsw - universe index for southwest quadrant of water rod
 - -     ncwrnw - cell index for water inside water rod for northwest
 - -              quadrant of water rod
*/
  short int n;
  int len = 132, length, index;
  int n_entries;
  int ufr, ufrl, uchan, uwrnw, uwrne, uwrse, uwrsw;
  int cchan, cwic;
```

```
  int ncpellet, ncgap, ncclad, ncwater, ncwrnw;
/* - Float Variable(s)
 - -    spor - reference fuel pellet outer surface
 - -    scir - reference cladding inner surface
 - -    scor - reference cladding outer surface
 - -    wror - radius for outer surface of water rod
 - -    wrir - radius for inner surface of water rod
 - - xminfrw - XMIN surface for fuel rod window
 - - xmaxfrw - XMAX surface for fuel rod window
 - - yminfrw - XMIN surface for fuel rod window
 - - ymaxfrw - YMAX surfacr for fuel rod window
 - -  wgcowx - wide gap, channel outside wall, px surface
 - -  wgciwx - wide gap, channel inside wall, px surface
 - -  ngciwx - narrow gap, channel inside wall, px surface
 - -  ngcowx - narrow gap, channel outside wall, px surface
 - -  wgcowy - wide gap, channel outside wall, py surface
 - -  wgciwy - wide gap, channel inside wall, py surface
 - -  ngciwy - narrow gap, channel inside wall, py surface
 - -  ngcowy - narrow gap, channel outside wall, py surface
 - - xambig1 - ambiguity surface for channel corners (wide gap)
 - - xambig2 - ambiguity surface for channel corners (narrow gap)
 - - yambig1 - ambiguity surface for channel corners (wide gap)
 - - yambig2 - ambiguity surface for channel corners (narrow gap)
 - -    ccro - outer radius for channel corner
 - -    ccri - innner radius for channel corner
 - -    cc1x - center for channel corner 1
 - -    cc1y - center for channel corner 1
 - -    cc2x - center for channel corner 2
 - -    cc2y - center for channel corner 2
 - -    cc3x - center for channel corner 3
 - -    cc3y - center for channel corner 3
 - -    cc4x - center for channel corner 4
 - -    cc4y - center for channel corner 4
 - - density - density for material
 - - x_ofset - x-offset for fuel rod centering to base lattice position
 - - y_ofset - y-offset for fuel rod centering to base lattice position
*/
  float spor, scir, scor,xminfrw,xmaxfrw,yminfrw,ymaxfrw,wgcowx,wgciwx
   ,ngciwx,ngcowx,wgcowy,wgciwy,ngciwy,ngcowy,xambig1,xambig2,yambig1
   ,yambig2,ccro,ccri,cc1x,cc1y,cc2x,cc2y,cc3x,cc3y,cc4x,cc4y;
  float density,x_ofset,y_ofset,wrir,wror;
/* - Character Variable(s)
 - -    label - label for cell or surface
 - - material - label for cell material for output edit
 - - mnemonic - MCNP mnemonic for surface definition
 - -    value - surface defintion values in string form
 - -   mdsnam - working pointer for lattice material intermediate
 - -           dataset name
*/
  char mnemonic[4];
  ascii_string label,material,value;
/* - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
```

```
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition linked list
*/
  ll_material *ptr_mtl;
/* - Compute Surfaces for Lattice - - - - - - - - - - - - - - - - - - - - */
  spor = pod/2;
  scir = (cod/2)-cld;
  scor = cod/2;
  wror = wrod/2;
  wrir = wror-wrth;
  xminfrw = -rpitch/2;
  xmaxfrw = rpitch/2;
  yminfrw = -rpitch/2;
  ymaxfrw = rpitch/2;
  wgcowx = wgap;
  wgciwx = wgap+cthick;
  ngciwx = wgap+cthick+asin;
  ngcowx = wgap+(2*cthick)+asin;
  wgcowy = -wgap;
  wgciwy = -(wgap+cthick);
  ngciwy = -(wgap+cthick+asin);
  ngcowy = -(wgap+(2*cthick)+asin);
  xambig1 = wgap+cthick+cradius;
  xambig2 = wgap+cthick+asin-cradius;
  yambig1 = -(wgap+cthick+cradius);
  yambig2 = -(wgap+cthick+asin-cradius);
  ccro = cradius+cthick;
  ccri = cradius;
  cc1x = wgap+cthick+cradius;
  cc1y = -(wgap+cthick+cradius);
  cc2x = wgap+cthick+asin-cradius;
  cc2y = -(wgap+cthick+cradius);
  cc3x = wgap+cthick+asin-cradius;
  cc3y = -(wgap+cthick+asin-cradius);
  cc4x = wgap+cthick+cradius;
  cc4y = -(wgap+cthick+asin-cradius);
/* - compute offset to move fuel rod to reference location */
  x_ofset = wgap+cthick+cfsrd+(cod/2);
  y_ofset = -x_ofset;
/* - Build Lattice Model Starting with Fuel Pellet - - - - - - - - - */
  for( n = 0; n < nft; n++){
    (*nuniverse)++;
    if(n == 0) ufr = *nuniverse;
/* - Pellet Cell */
    sprintf(label,"Fuel Pellet, #%i, L%i",(n+1),nlat);
    sprintf(material,"%s (%i)",matdsnam,(n+1));
    search_usage_list(1,material,&index,ptr_material_usage);
    if(n == 0){
      if(index == 0){
        (*nmaterial)++;
        add_cell(label,material,lu8,ncell,*nmaterial,-(*fp_density)
         ,nuniverse);
        n_entries = 1;}
      else{
        add_cell(label,material,lu8,ncell,index,-(*fp_density)
```

```
          ,nuniverse);
        n_entries = 0;}
      ncpellet = *ncell;}
    else{
      if(index == 0){
        (*nmaterial)++;
        sprintf(value,"mat= %i rho= %10.4E"
         ,*nmaterial,-(*fp_density));
        n_entries = 1;}
      else{
        sprintf(value,"mat= %i rho= %10.4E"
         ,index,-(*fp_density));
        n_entries = 0;}
      add_like_but(label,material,lu8,ncell,value,ncpellet
       ,nuniverse);}
    if(index == 0){
      { usage_list *ptr_ml = ptr_material_usage;
        while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
        index = (ptr_ml->index)+1;
        ptr_ml = load_usage_list(material,index,ptr_ml);
      }
    }
    fp_density++;
/* - Pellet Surface */
    if(n == 0){
      strcpy(label,"");
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i"
       ,(n+1),nlat);
      strcpy(mnemonic,"c/z");
      strcpy(value,"");
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,spor);
      index = -1;
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Pellet Materials */
    sprintf(label,"%s (%i)",matdsnam,(n+1));
/* - Determine number of entries in Linked List */
    if(n_entries == 1){
      { ll_material *f = fmids;
        do{
          f = f->next;
          n_entries++;}
        while(f->next != NULL);
      }
      add_material(lu10,nmaterial,label,n_entries,fmids);
    }
    if(n < (nft+1)) fmids++;
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Fuel/Cladding Gap - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Fuel/Cladding Gap, #%i, L%i",(n+1),nlat);
    strcpy(material,"void");
    if(n == 0){
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
```

```
      ncgap = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncgap,nuniverse);
/* - Surfaces */
    if(n == 0){
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge8x8_lattice_lcwr --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scir);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material -- void used */
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Cladding - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Cladding, #%i, L%i",(n+1),nlat);
    strcpy(material,frcmat);
    if(n == 0){
      search_usage_list(1,frcmat,&index,ptr_material_usage);
      n_entries = 0;
      if(index == 0){
        { usage_list *ptr_ml;
          ptr_mtl =
           material_match(ptr_core_mtls,frcmat,&density,&n_entries);
          (*nmaterial)++;
          ptr_ml = ptr_material_usage;
          while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
          index = (ptr_ml->index)+1;
          ptr_ml = load_usage_list(frcmat,index,ptr_ml);
        }
      }
      else{
        (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
        n_entries = 0;}
      add_cell(label,material,lu8,ncell,index,-density,nuniverse);
      ncclad = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncclad,nuniverse);
/* - Surface(s) */
    if(n == 0){
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
```

```
        ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge8x8_lattice_lcwr --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scor);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material */
    if(n == 0){
      if(n_entries != 0){
        add_material(lu10,nmaterial,frcmat,n_entries,ptr_mtl);
        rollup_llm(ptr_mtl);
        n_entries = 0;}
    }
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Water Outside of Outer Surface of Cladding - - - - - - - - - - -  */
/* - Cell */
    sprintf(label,"Water Outside Cladding #%i, L%i",(n+1),nlat);
    strcpy(material,inchannel_material);
    if(n == 0){
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge8x8_lattice_lcwr --\n");
        fprintf(nout
         ," Material Not Found in Linked List, material = %s\n"
         ,material);
        abort();}
      add_cell(label,material,lu8,ncell,index,-inchannel_density
       ,nuniverse);
      ncwater = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncwater,nuniverse);
/* - Surface */
    if(n == 0){
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","","",
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge8x8_lattice_lcwr --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
```

```
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
/* - Materials */
/* - In-channel Material should already exist as a material */
    if(n == 0) fprintf(lu8,"\n");
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
  }
/* - Create Water Rod for Lattice - - - - - - - - - - - - - - - - - - - - */
/* - Water Rod in Northwest Quadrant */
/* - Water */
  (*nuniverse)++;
  uwrnw = *nuniverse;
  sprintf(label,"Water Rod Center (NW Quadrant), L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density
   ,nuniverse);
  ncwrnw = *ncell;
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (NW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset+xmaxfrw)
     ,(y_ofset-ymaxfrw),wrir);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);
  }
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrnw);
/* - Cladding */
  sprintf(label,"Water Rod (NW Quadrant), L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
```

```
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (NW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Water Rod Outer Surface, (NW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset+xmaxfrw)
     ,(y_ofset-ymaxfrw),wror);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);}
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrnw);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod (NW Quadrant), L%i"
   ,nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Outer Surface, (NW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrnw);
/* - Water Rod Section in Northeast Quadrant */
/* - Inner Water */
  (*nuniverse)++;
  uwrne = *nuniverse;
  sprintf(label,"Water Rod Center (NE Quadrant), L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (NE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset-xmaxfrw)
     ,(y_ofset-ymaxfrw),wrir);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);
  }
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrne);
/* - Cladding */
  sprintf(label,"Water Rod (NE Quadrant), L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
```

```
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (NE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Water Rod Outer Surface, (NE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset-xmaxfrw)
     ,(y_ofset-ymaxfrw),wror);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);}
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrne);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod (NE Quadrant), L%i"
   ,nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Outer Surface, (NE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
```

```
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrne);
/* - Water Rod Section in Southeast Quadrant */
/* - Inner Water */
  (*nuniverse)++;
  uwrse = *nuniverse;
  sprintf(label,"Water Rod Center (SE Quadrant), L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (SE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset-xmaxfrw)
     ,(y_ofset+ymaxfrw),wrir);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);
  }
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrse);
/* - Cladding */
  sprintf(label,"Water Rod (SE Quadrant), L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (SE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
```

```
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Water Rod Outer Surface, (SE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset-xmaxfrw)
     ,(y_ofset+ymaxfrw),wror);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);}
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrse);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod (SE Quadrant), L%i"
   ,nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Outer Surface, (SE Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrse);
/* - Water Rod Section in Southwest Quadrant */
```

```
/* - Inner Water */
  (*nuniverse)++;
  uwrsw = *nuniverse;
  sprintf(label,"Water Rod Center (SW Quadrant), L%i",nlat);
    strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (SW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset+xmaxfrw)
     ,(y_ofset+ymaxfrw),wrir);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);
  }
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n      u= %i imp:n= 1.0\n",uwrsw);
/* - Cladding */
  sprintf(label,"Water Rod (SW Quadrant), L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Inner Surface, (SW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Water Rod Outer Surface, (SW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    strcpy(mnemonic,"c/z");
    sprintf(value,"%10.4E %10.4E %10.4E",(x_ofset+xmaxfrw)
     ,(y_ofset+ymaxfrw),wror);
    index = -1;
    add_surface(0,label,mnemonic,lu8,lu9,&index,value
     ,ptr_surface_usage,nsurface);}
  else{
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);}
  fprintf(lu8,"\n       u= %i imp:n= 1.0\n",uwrsw);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod (SW Quadrant), L%i"
   ,nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Water Rod Outer Surface, (SW Quadrant), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n       u= %i imp:n= 1.0\n",uwrsw);
/* - Create Lattice for Fuel Assembly and Populate with Previously
 - - Defined Fuel Rod Universes - - - - - - - - - - - - - - - - - -  */
/* - Cell */
  sprintf(label,"Reference Fuel Rod Cell, L%i",nlat);
  strcpy(material,inchannel_material);
```

```
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  (*nuniverse)++;
  ufrl = *nuniverse;
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"XMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",(xmaxfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"XMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",(xminfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",(yminfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",(ymaxfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8," lat= 1 u = %i imp:n= 1.0\n",*nuniverse);
  fprintf(lu8,"      fill= -1:%i -1:%i 0:0\n"
   ,latdim,latdim);
  lines(latdim+1);
  fprintf(nout,"     Universes Filling the Lattice:\n");
  { short int i,j;
    int nwr = 0, uwr;
    int *plattice = lattice;
    fprintf(lu8,"         ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
    for(i = 1;i <= latdim;i++){
      fprintf(lu8,"        %3i ",*nuniverse);
      fprintf(nout,"       ");
      for(j = 1;j <= latdim;j++,plattice++){
        if(*plattice > 0){
          fprintf(lu8,"%3i ",(*plattice+ufr-1));
          fprintf(nout,"%3i ",(*plattice+ufr-1));}
```

```
        else{
          switch (nwr) {
            case 0:
              uwr = uwrnw;
              break;
            case 1:
              uwr = uwrne;
              break;
            case 2:
              uwr = uwrsw;
              break;
            case 3:
              uwr = uwrse;
          }
          nwr++;
          fprintf(lu8,"%3i ",uwr);
          fprintf(nout,"%3i ",uwr);}
      }
      fprintf(lu8,"%3i\n",*nuniverse);
      fprintf(nout,"\n");}
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
  }
/* - Create Channel Model and Populate with Lattice - - - - - - - -  */
/* - Channel - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*nuniverse)++;
  uchan = *nuniverse;
  sprintf(label,"Channel, L%i",nlat);
  search_usage_list(1,fcmat,&index,ptr_material_usage);
  n_entries = 0;
  if(index == 0){
    { usage_list *ptr_ml;
      ptr_mtl =
       material_match(ptr_core_mtls,fcmat,&density,&n_entries);
      (*nmaterial)++;
      ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(fcmat,index,ptr_ml);
    }
  }
  else{
    (void) material_match(ptr_core_mtls,fcmat,&density,&n_entries);
    n_entries = 0;
  }
  add_cell(label,fcmat,lu8,ncell,index,-density,nuniverse);
  cchan = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = 1;
```

```
sprintf(value,"%10.4E",wgcowx);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",wgciwx);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",yambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",yambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,"):(");
sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",wgcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",wgciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = 1;
sprintf(value,"%10.4E",xambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",xambig2);
```

```
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,")\n");
fprintf(lu8,"      (");
sprintf(label,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",ngciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",ngcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,")(");
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
strcpy(mnemonic,"px");
sprintf(value,"%10.4E",ngciwx);
add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
 ,nsurface);
```

```
  sprintf(label
   ,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngcowx);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Channel Corners */
/* - Northwest Corner */
  sprintf(label,"Channel Corner Outer Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
```

```
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Northeast Corner */
  sprintf(label,"Channel Corner Outer Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
```

```
      ," Surface Not Found in Linked List, label = %s\n",label);
     abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
      ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"       (");
/* - Southeast Corner */
  sprintf(label,"Channel Corner Outer Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
      ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","",""
   ,ptr_surface_usage);
  if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Southwest Corner */
  sprintf(label,"Channel Corner Outer Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
```

```
  fprintf(lu8,")\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",uchan);
/* - Materials */
  if(n_entries != 0){
    add_material(lu10,nmaterial,fcmat,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
    n_entries = 0;}
/* - Area Inside the Channel - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Active Fuel Area, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density,nuniverse);
  cwic = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
```

```
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):(");
sprintf(label
 ,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8," ):\n      (");
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
```

```
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8,"):(");
 sprintf(label
  ,"Channel Corner Inner Radius (NW), L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge8x8_lattice_lcwr --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","",""
  ,ptr_surface_usage);
```

```
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,"):\n");
fprintf(lu8,"      (");
sprintf(label
 ,"Channel Corner Inner Radius (NE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
```

```
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,")");
sprintf(label
 ,"Channel Corner Inner Radius (SE), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","","",
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge8x8_lattice_lcwr --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
```

```
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
  sprintf(label
   ,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      fill= %i u= %i imp:n= 1.0\n",ufrl,uchan);
  lines(1);
  fprintf(nout,"      Filled with Universe %i\n",ufrl);
```

```
/* - Area Outside the Channel - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Water Outside of Channel, L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Regions Outside of Four Channel Walls */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
```

```
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
```

```
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
```

```
    ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
    ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge8x8_lattice_lcwr --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
/* - Assigned Universe Number to Return Variable */
  *ufl = uchan;
}
```

Function ge8x8_lattice_swr

```
#include <stdio.h>

#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
```

```
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                      } surface_usage_list;

typedef struct fuel_geometry{
                             int latdim;
                             int nwr;
                             float cthick;
                             float asin;
                             float wgap;
                             float ngap;
                             float cradius;
                             float fsrd;
                             float cfsrd;
                             float rpitch;
                             float cod;
                             float cld;
                             float pod;
                             char frcmat[6];
                             char fcmat[6];
                            } fg_list;


void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_material(FILE *,int *,char[],int,ll_material *);
ll_material *material_match(a_record *,char[],float *,int *);
usage_list *load_usage_list(char[],int,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
void abort();
void lines(int);
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
void add_like_but(char[],char[],FILE *,int *,char[],int,int *);

void ge8x8_lattice_swr(float cthick,float asin,float wgap,float ngap
 ,float cradius,float fsrd,float cfsrd,float rpitch,float cod
 ,float cld,float pod,char frcmat[],char fcmat[],int latdim,int nft
 ,int *lattice,int *ncell,int *nmaterial,FILE *lu8,FILE *lu9
 ,FILE *lu10,surface_usage_list *ptr_surface_usage
```

```
 ,usage_list *ptr_material_usage,float inchannel_density
 ,float bypass_density,int *nuniverse,float *fp_density
 ,ascii_string matdsnam,ll_material *fmids,a_record *ptr_core_mtls
 ,int nlat,int *ufl,int *nsurface,char inchannel_material []){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e 8 x 8 _ l a t t i c e _ s w r  *  Create Lattice Model for
 - -                                          GE 8x8 Lattice with Water
 - -                                          Rods the same Size as the
 - -                                          Fuel Rods
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     cthick - Channel Thickness (cm)                          (input)
 - -       asin - Inner Span of Channel (cm)                      (input)
 - -       wgap - Wide Gap Thickness (cm)                         (input)
 - -       ngap - Narrow Gap Thickness (cm)                       (input)
 - -    cradius - Inner Radius of Channel Corner (cm)             (input)
 - -       fsrd - Clad Surface to Clad Surface Separation         (input)
 - -              (cm)
 - -      cfsrd - Clad Surface to Inner Channel Surface           (input)
 - -              Separation (cm)
 - -     rpitch - Fuel Rod Pitch (cm)                             (input)
 - -        cod - Cladding Outer Diameter (cm)                    (input)
 - -        cld - Cladding Thickness (cm)                         (input)
 - -        pod - Fuel Pellet Outer Diameter (cm)                 (input)
 - -     frcmat - Material Identifier for Fuel Rod Cladding       (input)
 - -      fcmat - Material Identifier for Channel                 (input)
 - -     latdim - Lattice Dimensionality                          (input)
 - -        nft - Number of Unique Fuel Rod Types                 (input)
 - -    lattice - Fuel Rod Type Map                               (input)
 - -      ncell - number of MCNP cells                              (i&o)
 - -  nmaterial - number of MCNP materials                          (i&o)
 - -        lu8 - stream pointer to scratch file for cell defin- (input)
 - -              itions
 - -        lu9 - stream pointer to scratch file for surface      (input)
 - -              definitions
 - -       lu10 - stream pointer to scratch file for material     (input)
 - -              definitions
 - - ptr_surface_usage
 - -            - list of defined surfaces                       (input)
 - - ptr_material_usage
 - -            - list of defined materials                      (input)
 - - inchannel_density
 - -            - density for in-channel moderator (g/cc)        (input)
 - - bypass_density
 - -            - density for bypass moderator (g/cc)            (input)
 - -  nuniverse - number of MCNP "universes"                        (i&o)
 - - fp_density - density for fuel pellets                        (input)
 - -   matdsnam - name for fuel intermediate material dataset     (input)
 - -      fmids - linked list for fuel material composition       (input)
 - - ptr_core_mtls
 - -            - pointer to linked list containing core mat-    (input)
 - -              erial definitions
 - -       nlat - index for unique lattice for which to create    (input)
 - -              MCNP lattice model
 - -        ufl - index for universe that contains the entire     (input)
```

```
 - -                lattice representation
 - -   nsurface - total number of surfaces                        (input)
 - - inchannel_material
 - -            - mnemonic for inchannel moderator material      (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -         n - index for loops
 - -       len - length of ascii strings
 - -     index - utility variable for indices
 - - n_entries - number of entries in linked list for fuel material
 - -       ufr - universe index for first fuel rod
 - -      ufrl - universe index for fuel rod window lattice
 - -     uchan - universe index for channel and contents
 - -     cchan - cell index for channel
 - -      cwic - cell index for area inside channel
 - -  ncpellet - cell index for reference fuel pellet
 - -     ncgap - cell index for reference fuel/cladding gap
 - -    ncclad - cell index for reference cladding
 - -   ncwater - cell index for water outside of reference fuel rod
 - -       uwr - universe index for water rod
*/
  short int n;
  int len = 132, length, index;
  int n_entries;
  int ufr, ufrl, uchan, uwr;
  int cchan, cwic;
  int ncpellet, ncgap, ncclad, ncwater;
/* - Float Variable(s)
 - -     spor - reference fuel pellet outer surface
 - -     scir - reference cladding inner surface
 - -     scor - reference cladding outer surface
 - - xminfrw - XMIN surface for fuel rod window
 - - xmaxfrw - XMAX surface for fuel rod window
 - - yminfrw - XMIN surface for fuel rod window
 - - ymaxfrw - YMAX surfacr for fuel rod window
 - -  wgcowx - wide gap, channel outside wall, px surface
 - -  wgciwx - wide gap, channel inside wall, px surface
 - -  ngciwx - narrow gap, channel inside wall, px surface
 - -  ngcowx - narrow gap, channel outside wall, px surface
 - -  wgcowy - wide gap, channel outside wall, py surface
 - -  wgciwy - wide gap, channel inside wall, py surface
 - -  ngciwy - narrow gap, channel inside wall, py surface
 - -  ngcowy - narrow gap, channel outside wall, py surface
 - - xambig1 - ambiguity surface for channel corners (wide gap)
 - - xambig2 - ambiguity surface for channel corners (narrow gap)
 - - yambig1 - ambiguity surface for channel corners (wide gap)
 - - yambig2 - ambiguity surface for channel corners (narrow gap)
 - -     ccro - outer radius for channel corner
 - -     ccri - innner radius for channel corner
 - -     cc1x - center for channel corner 1
 - -     cc1y - center for channel corner 1
 - -     cc2x - center for channel corner 2
 - -     cc2y - center for channel corner 2
```

```
 - -    cc3x - center for channel corner 3
 - -    cc3y - center for channel corner 3
 - -    cc4x - center for channel corner 4
 - -    cc4y - center for channel corner 4
 - - density - density for material
 - - x_ofset - x-offset for fuel rod centering to base lattice position
 - - y_ofset - y-offset for fuel rod centering to base lattice position
*/
  float spor, scir, scor,xminfrw,xmaxfrw,yminfrw,ymaxfrw,wgcowx,wgciwx
   ,ngciwx,ngcowx,wgcowy,wgciwy,ngciwy,ngcowy,xambig1,xambig2,yambig1
   ,yambig2,ccro,ccri,cc1x,cc1y,cc2x,cc2y,cc3x,cc3y,cc4x,cc4y;
  float density,x_ofset,y_ofset;
/* - Character Variable(s)
 - -    label - label for cell or surface
 - - material - label for cell material for output edit
 - - mnemonic - MCNP mnemonic for surface definition
 - -    value - surface defintion values in string form
 - -   mdsnam - working pointer for lattice material intermediate
 - -           dataset name
*/
  char mnemonic[4];
  ascii_string label,material,value;
/* - FILE Variable(s)
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition linked list
*/
  ll_material *ptr_mtl;
/* - Compute Surfaces for Lattice - - - - - - - - - - - - - - - - - - - - */
  spor = pod/2;
  scir = (cod/2)-cld;
  scor = cod/2;
  xminfrw = -rpitch/2;
  xmaxfrw = rpitch/2;
  yminfrw = -rpitch/2;
  ymaxfrw = rpitch/2;
  wgcowx = wgap;
  wgciwx = wgap+cthick;
  ngciwx = wgap+cthick+asin;
  ngcowx = wgap+(2*cthick)+asin;
  wgcowy = -wgap;
  wgciwy = -(wgap+cthick);
  ngciwy = -(wgap+cthick+asin);
  ngcowy = -(wgap+(2*cthick)+asin);
  xambig1 = wgap+cthick+cradius;
  xambig2 = wgap+cthick+asin-cradius;
  yambig1 = -(wgap+cthick+cradius);
  yambig2 = -(wgap+cthick+asin-cradius);
  ccro = cradius+cthick;
  ccri = cradius;
  cc1x = wgap+cthick+cradius;
  cc1y = -(wgap+cthick+cradius);
  cc2x = wgap+cthick+asin-cradius;
```

```
  cc2y = -(wgap+cthick+cradius);
  cc3x = wgap+cthick+asin-cradius;
  cc3y = -(wgap+cthick+asin-cradius);
  cc4x = wgap+cthick+cradius;
  cc4y = -(wgap+cthick+asin-cradius);
/* - compute offset to move fuel rod to reference location */
  x_ofset = wgap+cthick+cfsrd+(cod/2);
  y_ofset = -x_ofset;
/* - Build Lattice Model Starting with Fuel Pellet - - - - - - - - - - */
  for( n = 0; n < nft; n++){
    (*nuniverse)++;
    if(n == 0) ufr = *nuniverse;
/* - Pellet Cell */
    sprintf(label,"Fuel Pellet, #%i, L%i",(n+1),nlat);
    sprintf(material,"%s (%i)",matdsnam,(n+1));
    search_usage_list(1,material,&index,ptr_material_usage);
    if(n == 0){
      if(index == 0){
        (*nmaterial)++;
        add_cell(label,material,lu8,ncell,*nmaterial,-(*fp_density)
         ,nuniverse);
        n_entries = 1;}
      else{
        add_cell(label,material,lu8,ncell,index,-(*fp_density)
         ,nuniverse);
        n_entries = 0;}
      ncpellet = *ncell;}
    else{
      if(index == 0){
        (*nmaterial)++;
        sprintf(value,"mat= %i rho= %10.4E"
         ,*nmaterial,-(*fp_density));
        n_entries = 1;}
      else{
        sprintf(value,"mat= %i rho= %10.4E"
         ,index,-(*fp_density));
        n_entries = 1;}
      add_like_but(label,material,lu8,ncell,value,ncpellet
       ,nuniverse);}
    { usage_list *ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(material,index,ptr_ml);
    }
    fp_density++;
/* - Pellet Surface */
    if(n == 0){
      strcpy(label,"");
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i"
       ,(n+1),nlat);
      strcpy(mnemonic,"c/z");
      strcpy(value,"");
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,spor);
      index = -1;
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
```

```
        ,ptr_surface_usage,nsurface);
    }
/* - Pellet Materials */
    sprintf(label,"%s (%i)",matdsnam,(n+1));
/* - Determine number of entries in Linked List */
    if(n_entries == 1){
      { ll_material *f = fmids;
        do{
          f = f->next;
          n_entries++;}
        while(f->next != NULL);
      }
      add_material(lu10,nmaterial,label,n_entries,fmids);
    }
    if(n < (nft+1)) fmids++;
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Fuel/Cladding Gap - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Fuel/Cladding Gap; #%i, L%i",(n+1),nlat);
    strcpy(material,"void");
    if(n == 0){
      add_cell(label,material,lu8,ncell,0,0.0,nuniverse);
      ncgap = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncgap,nuniverse);
/* - Surfaces */
    if(n == 0){
      sprintf(label,"Fuel Pellet Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scir);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
    }
/* - Material -- void used */
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Cladding - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Cladding, #%i, L%i",(n+1),nlat);
    strcpy(material,frcmat);
    if(n == 0){
      search_usage_list(1,frcmat,&index,ptr_material_usage);
      n_entries = 0;
```

```
      if(index == 0){
        { usage_list *ptr_ml;
          ptr_mtl =
           material_match(ptr_core_mtls,frcmat,&density,&n_entries);
          (*nmaterial)++;
          ptr_ml = ptr_material_usage;
          while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
          index = (ptr_ml->index)+1;
          ptr_ml = load_usage_list(frcmat,index,ptr_ml);
        }
      }
      else{
        (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
        n_entries = 0;}
      add_cell(label,material,lu8,ncell,index,-density,nuniverse);
      ncclad = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncclad,nuniverse);
/* - Surface(s) */
    if(n == 0){
      sprintf(label,"Fuel Cladding Inner Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      strcpy(mnemonic,"c/z");
      index = -1;
      sprintf(value,"%10.4E %10.4E %10.4E",x_ofset,y_ofset,scor);
      add_surface(0,label,mnemonic,lu8,lu9,&index,value
       ,ptr_surface_usage,nsurface);
     }
/* - Material */
    if(n == 0){
      if(n_entries != 0){
        add_material(lu10,nmaterial,frcmat,n_entries,ptr_mtl);
        rollup_llm(ptr_mtl);
        n_entries = 0;}
    }
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
/* - Water Outside of Outer Surface of Cladding - - - - - - - - - - - - */
/* - Cell */
    sprintf(label,"Water Outside Cladding #%i, L%i",(n+1),nlat);
    strcpy(material,inchannel_material);
    if(n == 0){
      search_usage_list(1,material,&index,ptr_material_usage);
      if(index == 0){
        lines(3);
```

```
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Material Not Found in Linked List, material = %s\n"
         ,material);
        abort();}
      add_cell(label,material,lu8,ncell,index,-inchannel_density
       ,nuniverse);
      ncwater = *ncell;}
    else
      add_like_but(label,material,lu8,ncell,"",ncwater,nuniverse);
/* - Surface */
    if(n == 0){
      sprintf(label,"Fuel Cladding Outer Surface, #%i, L%i",(n+1),nlat);
      search_surface_usage_list(1,label,&index,"","",""
       ,ptr_surface_usage);
      if(index == 0){
        lines(3);
        fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
        fprintf(nout," ge7x7_lattice --\n");
        fprintf(nout
         ," Surface Not Found in Linked List, label = %s\n",label);
        abort();}
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
/* - Materials */
/* - In-channel Material should already exist as a material */
    if(n == 0) fprintf(lu8,"\n");
    fprintf(lu8,"      u= %i imp:n= 1.0\n",(ufr+n));
  }
/* - Create Water Rod for Lattice - - - - - - - - - - - - - - - - - - - - */
/* - Water */
  (*nuniverse)++;
  uwr = *nuniverse;
  sprintf(label,"Water Rod Center, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Fuel Cladding Inner Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
```

```
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Cladding */
  sprintf(label,"Water Rod Cladding, L%i",nlat);
  strcpy(material,frcmat);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Fuel Cladding Inner Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label,"Fuel Cladding Outer Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Water Outside of Outer Surface of Water Rod */
  sprintf(label,"Water Outside of Water Rod, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** Function");
```

```
    fprintf(nout," ge8x8_lattice_swr --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n"
     ,material);
    abort();}
  (void) material_match(ptr_core_mtls,frcmat,&density,&n_entries);
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"Fuel Cladding Outer Surface, #1, L%i",nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," u= %i imp:n= 1.0\n",uwr);
/* - Create Lattice for Fuel Assembly and Populate with Previously
 - - Defined Fuel Rod Universes - - - - - - - - - - - - - - - - - - -  */
/* - Cell */
  sprintf(label,"Reference Fuel Rod Cell, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  (*nuniverse)++;
  ufrl = *nuniverse;
  add_cell(label,material,lu8,ncell,index,-inchannel_density
   ,nuniverse);
/* - Surfaces */
  sprintf(label,"XMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",(xmaxfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"XMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",(xminfrw+x_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMIN Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = 1;
```

```
  sprintf(value,"%10.4E",(yminfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"YMAX Surface for Fuel Rod Window, L%i",nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",(ymaxfrw+y_ofset));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8," lat= 1 u = %i imp:n= 1.0\n",*nuniverse);
  fprintf(lu8,"      fill= -1:%i -1:%i 0:0\n"
   ,latdim,latdim);
  lines(latdim+1);
  fprintf(nout,"     Universes Filling the Lattice:\n");
  { short int i,j;
    int *plattice = lattice;
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
    for(i = 1;i <= latdim;i++){
      fprintf(lu8,"        %3i ",*nuniverse);
      fprintf(nout,"      ");
      for(j = 1;j <= latdim;j++,plattice++){
        if(*plattice > 0){
          fprintf(lu8,"%3i ",(*plattice+ufr-1));
          fprintf(nout,"%3i ",(*plattice+ufr-1));}
        else{
          fprintf(lu8,"%3i ",uwr);
          fprintf(nout,"%3i ",uwr);}
      }
      fprintf(lu8,"%3i\n",*nuniverse);
      fprintf(nout,"\n");}
    fprintf(lu8,"        ");
    for(j = 0;j <= (latdim+1);j++) fprintf(lu8,"%3i ",*nuniverse);
    fprintf(lu8,"\n");
  }
/* - Create Channel Model and Populate with Lattice - - - - - - - - - */
/* - Channel - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  (*nuniverse)++;
  uchan = *nuniverse;
  sprintf(label,"Channel, L%i",nlat);
  search_usage_list(1,fcmat,&index,ptr_material_usage);
  n_entries = 0;
  if(index == 0){
    { usage_list *ptr_ml;
      ptr_mtl =
       material_match(ptr_core_mtls,fcmat,&density,&n_entries);
      (*nmaterial)++;
      ptr_ml = ptr_material_usage;
      while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
      index = (ptr_ml->index)+1;
      ptr_ml = load_usage_list(fcmat,index,ptr_ml);
    }
  }
```

```
  else{
    (void) material_match(ptr_core_mtls,fcmat,&density,&n_entries);
    n_entries = 0;
  }
  add_cell(label,fcmat,lu8,ncell,index,-density,nuniverse);
  cchan = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = 1;
  sprintf(value,"%10.4E",wgcowx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  index = -1;
  sprintf(value,"%10.4E",wgciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",yambig1);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",yambig2);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,") : (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = -1;
  sprintf(value,"%10.4E",wgcowy);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"py");
  index = 1;
  sprintf(value,"%10.4E",wgciwy);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
```

```
 ,nlat);
strcpy(mnemonic,"px");
index = 1;
sprintf(value,"%10.4E",xambig1);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
strcpy(mnemonic,"px");
index = -1;
sprintf(value,"%10.4E",xambig2);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
fprintf(lu8,")\n");
fprintf(lu8,"      (");
sprintf(label,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
 ,nlat);
strcpy(mnemonic,"py");
index = -1;
sprintf(value,"%10.4E",ngciwy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
,nlat);
strcpy(mnemonic,"py");
index = 1;
sprintf(value,"%10.4E",ngcowy);
add_surface(0,label,mnemonic,lu8,lu9,&index,value
 ,ptr_surface_usage,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
```

```
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngciwx);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  strcpy(mnemonic,"px");
  sprintf(value,"%10.4E",ngcowx);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value,ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Channel Corners */
/* - Northwest Corner */
  sprintf(label,"Channel Corner Outer Radius (NW), L%i"
   ,nlat);
```

```
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc1x,cc1y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Northeast Corner */
  sprintf(label,"Channel Corner Outer Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc2x,cc2y,ccri);
```

```
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"      (");
/* - Southeast Corner */
  sprintf(label,"Channel Corner Outer Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SE), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc3x,cc3y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
/* - Southwest Corner */
  sprintf(label,"Channel Corner Outer Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = -1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccro);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label,"Channel Corner Inner Radius (SW), L%i"
   ,nlat);
  strcpy(mnemonic,"c/z");
  index = 1;
  sprintf(value,"%10.4E %10.4E %10.4E",cc4x,cc4y,ccri);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
```

```
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",uchan);
/* - Materials */
  if(n_entries != 0){
    add_material(lu10,nmaterial,fcmat,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
    n_entries = 0;}
/* - Area Inside the Channel - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell */
  sprintf(label,"Active Fuel Area, L%i",nlat);
  strcpy(material,inchannel_material);
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-inchannel_density,nuniverse);
  cwic = *ncell;
/* - Surfaces */
  fprintf(lu8," (");
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
```

```
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Wide Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Narrow Gap, Channel Inside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):(");
  sprintf(label
   ,"Wide Gap, Channel Inside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
```

```
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8," ):\n      (");
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
```

```
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Narrow Gap, Channel Inside Wall, pX Surface, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
index = -index;
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
sprintf(label
 ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
  fprintf(nout," ge7x7_lattice --\n");
  fprintf(nout
   ," Surface Not Found in Linked List, label = %s\n",label);
  abort();}
add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
 ,nsurface);
fprintf(lu8,")");
sprintf(label
 ,"Channel Corner Inner Radius (NW), L%i"
 ,nlat);
search_surface_usage_list(1,label,&index,"","",""
 ,ptr_surface_usage);
if(index == 0){
  lines(3);
  fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
```

```
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"):\n");
  fprintf(lu8,"       (");
  sprintf(label
   ,"Channel Corner Inner Radius (NE), L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
```

```
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8,"):(");
 sprintf(label
  ,"Channel Corner Inner Radius (SE), L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 fprintf(lu8,"):\n");
 fprintf(lu8,"      (");
 sprintf(label
  ,"Channel Corner Inner Radius (SW), L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
   fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
   fprintf(nout," ge7x7_lattice --\n");
   fprintf(nout
    ," Surface Not Found in Linked List, label = %s\n",label);
   abort();}
 index = -index;
 add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
  ,nsurface);
 sprintf(label
  ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
  ,nlat);
 search_surface_usage_list(1,label,&index,"","","""
  ,ptr_surface_usage);
 if(index == 0){
   lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,")\n");
  fprintf(lu8,"      fill= %i u= %i imp:n= 1.0\n",ufrl,uchan);
  lines(1);
  fprintf(nout,"     Filled with Universe %i\n",ufrl);
/* - Area Outside the Channel - - - - - - - - - - - - - - - - - - - -   */
/* - Cell */
  sprintf(label,"Water Outside of Channel, L%i",nlat);
  strcpy(material,"Bypass Water");
  search_usage_list(1,material,&index,ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Material Not Found in Linked List, material = %s\n",label);
    abort();}
  add_cell(label,material,lu8,ncell,index,-bypass_density,nuniverse);
/* - Surfaces */
/* - Regions Outside of Four Channel Walls */
  fprintf(lu8," (");
  sprintf(label,"Wide Gap, Channel Outside Wall, pX Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Wide Gap, Channel Outside Wall, pY Surface, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
```

```
    ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pX Surface, L%i"
    ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
    ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
      ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
    ,nsurface);
  fprintf(lu8," : ");
  sprintf(label,"Narrow Gap, Channel Outside Wall, pY Surface, L%i"
    ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
    ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
      ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
    ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NW Corner */
  fprintf(lu8,"       (");
  sprintf(label,"Channel Corner Outer Radius (NW), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
    ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
      ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
    ,nsurface);
  sprintf(label
    ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
    ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
    ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
```

```
      ," Surface Not Found in Linked List, label = %s\n",label);
     abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- NE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (NE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SE Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SE), L%i",nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," ):\n");
/* - Regions Outside of Channel Outside Corners -- SW Corner */
  fprintf(lu8,"      (");
  sprintf(label,"Channel Corner Outer Radius (SW), L%i",nlat);
```

```
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Wide Gap), pX, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  sprintf(label
   ,"Ambiguity Surface for Channel Corners (Narrow Gap), pY, L%i"
   ,nlat);
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  ***  Function");
    fprintf(nout," ge7x7_lattice --\n");
    fprintf(nout
     ," Surface Not Found in Linked List, label = %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8," )\n");
  fprintf(lu8,"      u= %i imp:n= 1.0\n",*nuniverse);
/* - Assigned Universe Number to Return Variable */
  *ufl = uchan;
}
```

Function generate_lattice_model

```
#include <stdio.h>
#include <string.h>
typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
```

```
                                } a_record;
typedef struct s_material{
                                struct s_material *last;
                                int atomic_number;
                                int mass_number;
                                float weight_percentage;
                                char library_suffix[5];
                                struct s_material *next;
                               } ll_material;
typedef struct u_list {
                            struct u_list *last;
                            int index;
                            ascii_string label;
                            struct u_list *next;
                          } usage_list;

typedef struct su_list {
                             struct su_list *last;
                             int index;
                             ascii_string label;
                             ascii_string value;
                             char mnemonic[4];
                             ascii_string equivalent_label;
                             struct su_list *next;
                           } surface_usage_list;

typedef struct fuel_geometry {
                                  ascii_string gds_name;
                                  int latdim;
                                  int nwr;
                                  float cthick;
                                  float asin;
                                  float wgap;
                                  float ngap;
                                  float cradius;
                                  float fsrd;
                                  float cfsrd;
                                  float rpitch;
                                  float cod;
                                  float cld;
                                  float pod;
                                  float wrod;
                                  float wrth;
                                  char frcmat[6];
                                  char fcmat[6];
                                } fg_list;


void ge7x7_lattice(float,float,float,float,float,float,float,float
 ,float,float,float,char[],char[],int,int,int *,int *,int *,FILE *
 ,FILE *,FILE *,surface_usage_list *,usage_list *,float,float,int *
 ,float *,ascii_string,ll_material *,a_record *,int,int *,int *
 ,char[]);
void ge8x8_lattice_swr(float,float,float,float,float,float,float,float
 ,float,float,float,char[],char[],int,int,int *,int *,int *,FILE *
```

```
 ,FILE *,FILE *,surface_usage_list *,usage_list *,float,float,int *
 ,float *,ascii_string,ll_material *,a_record *,int,int *,int *
 ,char[]);
void ge8x8_lattice_lcwr(float,float,float,float,float,float,float,float
 ,float,float,float,char[],char[],int,int,int *,int *,int *,FILE *
 ,FILE *,FILE *,surface_usage_list *,usage_list *,float,float,int *
 ,float *,ascii_string,ll_material *,a_record *,int,int *,int *
 ,char[],float,float,int *);
void rollup_llm(ll_material *);
int *memory_integer(int,int,int *);
ll_material *load_fuel_material(int *,char[],char[],int,int **
 ,float **);
int mchar(int *,char[]);

void generate_lattice_model(int *ncell,int *nmaterial,FILE *lu8
 ,FILE *lu9,FILE *lu10,surface_usage_list *ptr_surface_usage
 ,usage_list *ptr_material_usage,int *ptr_correspondence_table
 ,fg_list *ptr_lg_ds,ascii_string *lmdsnam,int nlat,char fprefix[]
 ,float inchannel_density,float bypass_density,int *nuniverse
 ,a_record *ptr_core_mtls,int *ufl,int *nsurface
 ,char inchannel_material[],int *ncell_tr){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  g e n e r a t e _ l a t t i c e _ m o d e l  *  Create Lattice
 - -                                                   Model for a
 - -                                                   Unique Node
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      ncell - number of MCNP cells                                (i&o)
 - -  nmaterial - number of MCNP materials                           (i&o)
 - -        lu8 - stream pointer to scratch file for cell defin-  (input)
 - -              itions
 - -        lu9 - stream pointer to scratch file for surface      (input)
 - -              definitions
 - -       lu10 - stream pointer to scratch file for material     (input)
 - -              definitions
 - - ptr_surface_usage
 - -            - list of defined surfaces                       (input)
 - - ptr_material_usage
 - -            - list of defined materials                      (input)
 - - ptr_correspondence_table
 - -            - table relating lattices with unique material   (input)
 - -              definitions with geometry definitions
 - -  ptr_lg_ds - list of lattice geometry datasets              (input)
 - -    lmdsnam - list of material intermediate dataset names    (input)
 - -       nlat - index for unique lattice for which to create   (input)
 - -              MCNP lattice model
 - -    fprefix - prefix for fuel material intermediate datasets (input)
 - - inchannel_density
 - -            - density for in-channel moderator (g/cc)        (input)
 - - bypass_density
 - -            - density for bypass moderator (g/cc)            (input)
 - -  nuniverse - number of MCNP "universes"                        (i&o)
 - - ptr_core_mtls
 - -            - pointer to linked list containing core mat-    (input)
 - -              erial definitions
```

```
 - -        ufl - universe definition for unique lattices        (output)
 - -   nsurface - total number of surfaces                          (i&o)
 - - inchannel_material
 - -             - menmonic for inchannel moderator material      (input)
 - -   ncell_tr - number of MCNP cells for translated cells         (i&o)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -  latdim - lattice dimensionality
 - -     nwr - number of water rods
 - -   nglat - index for lattice geometry dataset corresponding to
 - -           lattice material dataset
 - - lattice - fuel rod type map for lattice
 - -     nft - number of fuel rod types
*/
  int latdim, nwr;
  int nglat, *lattice, nft;
/* - Float Variable(s)
 - -     cthick - Channel Thickness (cm)
 - -       asin - Inner Span of Channel (cm)
 - -       wgap - Wide Gap Thickness (cm)
 - -       ngap - Narrow Gap Thickness (cm)
 - -    cradius - Inner Radius of Channel Corner (cm)
 - -       fsrd - Clad Surface to Clad Surface Separation (cm)
 - -      cfsrd - Clad Surface to Inner Channel Surface
 - -              Separation (cm)
 - -     rpitch - Fuel Rod Pitch (cm)
 - -        cod - Cladding Outer Diameter (cm)
 - -        cld - Cladding Thickness (cm)
 - -        pod - Fuel Pellet Outer Diameter (cm)
 - -       wrod - Water Rod Outer Diameter (cm)
 - -       wrth - Water Rod Thickness (cm)
 - - fp_density - Density Vector for Lattice
*/
  float cthick, asin, wgap, ngap, cradius, fsrd, cfsrd, rpitch
   ,cod, cld, pod, wrod, wrth;
  float *fp_density;
/* - Character Datasets
 - -  matdsnam - name for material dataset
 - -    frcmat - Material Identifier for Fuel Rod Cladding
 - -     fcmat - Material Identifier for Channel
 - -       gds - name for lattice geometry dataset
*/
  char frcmat[6], fcmat[6];
  ascii_string matdsnam, gds;
/* - Structured Variable(s)
 - -  ptr_fg - pointer to proper fuel material dataset
 - -   fmids - pointer to fuel material dataset
*/
  fg_list *ptr_fg = ptr_lg_ds;
  ll_material *fmids;
/* - Adjust Character String Read with FORTRAN for Processing in C -  */
  { short int n;
    int len = 132, length;
```

```
    ascii_string *p = lmdsnam;
    for( n = 1; n < nlat; n++,p++) continue;
    strncpy(matdsnam,*p,132);
    length = mchar(&len,matdsnam);
    matdsnam[length] = '\0';
  }
/* - Find Corresponding Lattice Geometry Dataset */
  { short int n;
    int *p = ptr_correspondence_table;
/*    do{
      p++;p++;}
    while(*p != nlat);
*/
    while(*p != nlat) {p++;p++;}
    nglat = *(p+1);
    for( n = 0; n < (nglat-1); n++) ptr_fg++;
    latdim = ptr_fg->latdim;
    strcpy(gds,ptr_fg->gds_name);
  }
/* - Allocate Memory for Lattice Map */
  lattice = memory_integer(1,(latdim*latdim),lattice);
/* - Load Material Dataset for Lattice - - - - - - - - - - - - - - - - - - */
  fmids = load_fuel_material(&nft,fprefix,matdsnam,latdim,&lattice
   ,&fp_density);
/* - Create Lattice Model - - - - - - - - - - - - - - - - - - - - - - - - - */
   cthick = ptr_fg->cthick;    asin = ptr_fg->asin;
     wgap = ptr_fg->wgap;      ngap = ptr_fg->ngap;
  cradius = ptr_fg->cradius;   fsrd = ptr_fg->fsrd;
    cfsrd = ptr_fg->cfsrd;   rpitch = ptr_fg->rpitch;
      cod = ptr_fg->cod;        cld = ptr_fg->cld;
      pod = ptr_fg->pod;
     wrod = ptr_fg->wrod;      wrth = ptr_fg->wrth;
  strcpy(frcmat,ptr_fg->frcmat);
  strcpy(fcmat,ptr_fg->fcmat);
  if(strstr(gds,"ge7x7") != NULL)
    ge7x7_lattice(cthick,asin,wgap,ngap,cradius,fsrd,cfsrd,rpitch,cod
     ,cld,pod,frcmat,fcmat,latdim,nft,lattice,ncell,nmaterial,lu8,lu9
     ,lu10,ptr_surface_usage,ptr_material_usage,inchannel_density
     ,bypass_density,nuniverse,fp_density,matdsnam,fmids,ptr_core_mtls
     ,nlat,ufl,nsurface,inchannel_material);
   else
    if(wrod == 0.0)
      ge8x8_lattice_swr(cthick,asin,wgap,ngap,cradius,fsrd,cfsrd,rpitch,cod
       ,cld,pod,frcmat,fcmat,latdim,nft,lattice,ncell,nmaterial,lu8,lu9
       ,lu10,ptr_surface_usage,ptr_material_usage,inchannel_density
       ,bypass_density,nuniverse,fp_density,matdsnam,fmids,ptr_core_mtls
       ,nlat,ufl,nsurface,inchannel_material);
     else
      ge8x8_lattice_lcwr(cthick,asin,wgap,ngap,cradius,fsrd,cfsrd,rpitch,cod
       ,cld,pod,frcmat,fcmat,latdim,nft,lattice,ncell,nmaterial,lu8,lu9
       ,lu10,ptr_surface_usage,ptr_material_usage,inchannel_density
       ,bypass_density,nuniverse,fp_density,matdsnam,fmids,ptr_core_mtls
       ,nlat,ufl,nsurface,inchannel_material,wrod,wrth,ncell_tr);
/* - Return Memory */
  { short int n;
```

```
    ll_material *p = fmids, *p_last;
    for( n = 0; n < (nft-1); n++,p++) continue;
    p_last = p;
    for( n = nft; n > 0; n--){
      p = p_last;
      p_last--;
      rollup_llm(p);}
  }
  lattice = memory_integer(-1,(latdim*latdim),lattice);
}
```

Function material_match

```
#include<stdio.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

ll_material *memory_s_material(int,int,ll_material *);


ll_material *material_match(a_record *p,char name[],float *density
 ,int *n_entries){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * m a t e r i a l _ m a t c h  *  Matches Material Identifiers
 - -                                    with Materials on Linked List
 - -                                    for Core Materials
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -          p - pointer to first element in linked list        (input)
 - -       name - string containing name of material sought      (input)
 - -    density - density of material                            (output)
 - -  n_entries - number of elements/isotopes of which the        (output)
 - -              material is composed
 - -    m_first - pointer to first element in material defin-    (output)
 - -              ition link list
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -           n - counter for processing elements within material
 - -               definition
 - -  n_elements - number of elements of which a material is composed
 - -  n_isotopes - number of isotopes for a particular element in a
```

```
 - -                 material
 - -    a_number - atomic number from material in core materials
 - -                 linked list
 - -    m_number - mass number of isotope
*/
  short int n;
  int n_elements, n_isotopes, a_number, m_number;
/* - Float Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - -   ewp - elemental weight-percentage from core materials linked
 - -         list
 - -   iwp - isotopic weight-percentage from core materials linked
 - -         list
*/
  float ewp, iwp;
/* - Character Variable(s)
 - -  db_name - string to hold name of material from linked list
 - -     line - buffer to hold "line" from linked list
 - - l_suffix - library suffix from element or isotope on core mater-
 - -            ial linked list
 - -  element - elemental name from element on core material linked
 - -            list
*/
  char db_name[6], l_suffix[5], element[15];
  ascii_string line;
/* - FILE Variable(s) - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  nout - output file
*/
  extern FILE *nout;
/* - Structure Variable(s) - - - - - - - - - - - - - - - - - - - - - - - -
 - -  pm_current - pointer to current position in linked list for
 - -               material definition
 - -     pm_next - pointer to next position in linked list for material
 - -               definition
 - -     m_first - first element in linked list to material definition
*/
  ll_material *pm_current, *pm_next, *m_first = NULL;
/* - Sweep through the linked list seeking material identifier match */
  while(p->next != 0){
    strcpy(line,p->line);
    if(strstr(line,name) != NULL){          /* - match on identifier */
      *n_entries = 0;
      sscanf(line,"%s %f %i",db_name,density,&n_elements);
      for(n = 1; n <= n_elements; n++){     /* - sweep through ele - */
        p = p->next;                         /* - ment definitions -  */
        sscanf(p->line,"%s %f %i %s %i"
         ,element,&ewp,&a_number,l_suffix,&n_isotopes);
        pm_next = memory_s_material(1,1,pm_next);
        if(m_first == NULL) {
          m_first = pm_next;
          pm_current = pm_next;
          pm_current->last = NULL;}
        else{
          pm_current->next = pm_next;
          pm_next->last = pm_current;
          pm_current = pm_next;
```

```
        }
        pm_current->next = NULL;
        if(strlen(l_suffix) == 1){
          { short int i;
            *n_entries += n_isotopes;
            for(i = 1;i <= n_isotopes;i++){
              p = p->next;
              sscanf(p->line,"%i %f %s",&m_number,&iwp,l_suffix);
              iwp = ewp*iwp/100.0;
              pm_current->atomic_number = a_number;
              pm_current->mass_number = m_number;
              pm_current->weight_percentage = iwp;
              strcpy(pm_current->library_suffix,l_suffix);
              if(i < n_isotopes){
                pm_next = memory_s_material(1,1,pm_next);
                pm_current->next = pm_next;
                pm_next->last = pm_current;
                pm_current = pm_next;}
            }
          }}
        else{
          (*n_entries)++;
          pm_current->atomic_number = a_number;
          pm_current->mass_number = 0;
          pm_current->weight_percentage = ewp;
          strcpy(pm_current->library_suffix,l_suffix);
          { short int i;
            for(i = 1;i <= n_isotopes; i++) p=p->next;}
        }
      }
      return m_first;
    }
    p = p->next;
  }
/* - No match found -- fatal error - - - - - - - - - - - - - - - - - - - - */
  lines(4);
  fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
  fprintf(nout," material match:\n");
  fprintf(nout,"0No Match Found for Input Material %s\n",name);
  abort();
}
```

Function source_specification

```
#include <stdio.h>

void source_specification(int nsrck,float rkk,int ikz,int kct,FILE *lu10
 ,int ncolcc,int nrowcc,int *ccmap,int *ccmapw,float apitch,float afl
 ,int core_f,int ndm,int mct,int ndmp,int ncell,float tempk){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - *  s o u r c e _ s p e c i f i c a t i o n  *  Adds Source Specif-
 - -                                                ication to Input
 - -                                                Deck
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -      nsrck - nominal source size per cycle in MCNP          (input)
```

```
 - -        rkk - initial guess for eigenvalue in MCNP            (input)
 - -        ikz - number of cycles to be skipped before begin-    (input)
 - -              ning tally accumulation in MCNP
 - -        kct - number of cycles to be performed in MCNP        (input)
 - -       lu10 - stream pointer for material scratch file        (input)
 - -     ncolcc - Number of Columns in Control Cell Map           (input)
 - -     nrowcc - Number of Rows in Control Cell Map              (input)
 - -      ccmap - Map of Control Cell Universe Number             (input)
 - -     ccmapw - Map Containing Contents of Each Control Cell    (input)
 - -     apitch - fuel assembly pitch (cm)                        (input)
 - -        afl - active fuel length (cm)                         (input)
 - -     core_f - fraction of core modeled in problem             (input)
 - -              (0 - full,
 - -               1 - half,
 - -               4 - quarter)
 - -        ndm - periodicity of dumps to TPE file                (input)
 - -        mct - flag to creation of MCTAL file                  (input)
 - -       ndmp - maximum number of dumps on file                 (input)
 - -      ncell - total number of cells in the problem            (input)
 - -      tempk - problem temperature (Kelvin)                    (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Declarations - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        i - loop variable for columns in map
 - -        j - loop variable for rows in map
 - -        n - utility loop variable
 - -     nsrc - number of source fields
 - -  ptr_map - pointer to ccmap array
 - - ptr_mapw - pointer to ccmapw array
 - -   i_zero - column for origin of problem
 - -   j_zero - row of origin of problem
*/
  short int i,j,n,i_zero,j_zero;
  int nsrc = 0;
  int *ptr_map = ccmap, *ptr_mapw = ccmapw;
/* - Float Variable(s)
 - -   x - abscissa for center of source body
 - -   y - ordinate for center of source body
*/
  float x,y;
/* - Print Program Control and Source Cards - - - - - - - - - - - - - - - */
/* - The thermal scattering kernal temperature in MeV is computed from*/
/* - the formula given on page 3-108 of the MCNP-4B User's Manual - - */
  fprintf(lu10,"c     Control Cards\n");
  fprintf(lu10,"c\n");
  fprintf(lu10," tmp1 %10.4E %ir\n",(8.617E-11*tempk),(ncell-1));
  fprintf(lu10,"kcode  %5i %5.3f %3i %5i\n",nsrck,rkk,ikz,kct);
  fprintf(lu10,"print 40\n");
  fprintf(lu10," lost 20 20 \n");
  fprintf(lu10,"prdmp %i %i %i %i\n",kct,ndm,mct,ndmp);
/* - Source Fields */
  fprintf(lu10," sdef  erg=d1 pos=d2 axs= 0 0 1 rad=d3 ext=d4\n");
  fprintf(lu10,"  sp1  -3\n");
  fprintf(lu10,"  si2  L");
```

```
  for( j = 1; j <= nrowcc; j++){
    for( i = 1; i <= ncolcc; i++,ptr_map++){
/* - Check for Null Control Cell */
      if(*ptr_map == 0){
        for(n = 1;n <= 6; n++) ptr_mapw++;
        continue;}
/* - Compute Coordinates of Source Body Center */
      switch (core_f){
      case 4:
        i_zero = ncolcc;
        j_zero = nrowcc;
        break;
      case 1:
        i_zero = ncolcc/2;
        j_zero = nrowcc/2;
        break;
      case 2:
        i_zero = ncolcc;
        j_zero = nrowcc/2;
        break;
      }
      if((i == i_zero) || (j == j_zero)) continue;
      nsrc++;
      x = (i-i_zero)*2*apitch;
      y = (j_zero-j)*2*apitch;
      if(nsrc == 1)
        fprintf(lu10," %10.4E %10.4E %10.4E\n",x,y,(afl/2));
      else
        fprintf(lu10,"          %10.4E %10.4E %10.4E\n",x,y
         ,(afl/2));
    }
  }
  fprintf(lu10,"  sp2");
  { int nmax=10,nmin=1;
    do{
      for(n = nmin;n <= nmax;n++)
        fprintf(lu10," 1");
      fprintf(lu10,"\n");
      if(n <= nsrc) fprintf(lu10,"      ");
      nmin += 10;
      nmax += 10;
      if(nmax > nsrc) nmax = nsrc;
    }while(n <= nsrc);
  }
  fprintf(lu10,"  si3 %10.4E\n",((2*apitch)-0.01));
  fprintf(lu10,"  si4 %10.4E\n",(afl/2));
}
```

Function spacer_location

```
void spacer_location(int nbundlg,int *ptr_n_spacer
 ,float *ptr_spacer_location,int *ptr_spacer_node,int *ptr_n_node
 ,float *ptr_node_boundaries){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  *  s p a c e r _ l o c a t i o n  *  Determines Nodal Locations
 - -                                       of each Spacer
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -    nbundlg - number of unique fuel assembly geometries     (input)
 - - ptr_n_spacer
 - -            - number of spacers in each fuel assembly type (input)
 - - ptr_spacer_location
 - -            - pointer to array containing location of spa- (input)
 - -              cers for each fuel assembly type (cm)
 - - ptr_spacer_node
 - -            - pointer to array containing space node       (output)
 - -              locations for each fuel assembly type
 - - ptr_n_node - number of node boundaries for each unique     (input)
 - -              fuel assembly type
 - - ptr_node_boundaries
 - -            - axial locations of node boundaries for each  (input)
 - -              unique fuel assembly type
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -        i - utility index variable
 - -        j - utility index variable
 - -        k - utility index variable
 - -        n - utility index variable
 - -    ptr_n - pointer to integer vector
 - -   ptr_sn - pointer to integer vector containing spacer nodal
 - -            locations
 - - ptr_node - pointer to integer vector containing the number of
 - -            nodes for each unique fuel assembly type
 - -    found - flag to indicate that a node has been found in which
 - -            a spacer is resident
*/
  short int i,j,k,n,found;
  int *ptr_n = ptr_n_spacer, *ptr_sn = ptr_spacer_node;
  int *ptr_node = ptr_n_node;
/* - Float Variable(s)
 - -       ptr_f - pointer to floating point array
 - -      ptr_nb - pointer to floating point vector containing
 - -               the node boundaries
*/
  float *ptr_f = ptr_spacer_location;
  float *ptr_nb = ptr_node_boundaries;
/* - Sweep over the Unique Fuel Assembly Geometry Types - - - - - -  */
  for(i = 1;i <= nbundlg;i++,ptr_n++,ptr_node++){
/* - Loop over the Spacers in this Geometry Type */
    for(j = 1;j <= *ptr_n;j++,ptr_f++){
/* - Loop over the Axial Nodes in this Fuel Assembly Type */
      found = 0;
      ptr_nb = ptr_node_boundaries;
      for(n = 1;n < i;n++)
        for(k = 1;k <= *ptr_node;k++) *ptr_nb++;
      for(k = 1;k <= *ptr_node;k++,ptr_nb++){
        if((*ptr_f <= *ptr_nb) && (found != 1)){
          *ptr_sn = k;
          ptr_sn++;
```

```
            found = 1;}
        }
        if(found != 1){
          *ptr_sn = (*ptr_node)+1;
          ptr_sn++;}
      }
    }
}
```

Function vessel_generation

```
#include <stdio.h>
#include <string.h>

typedef char ascii_string[133];
typedef struct ascii_record{
                              struct ascii_record *last;
                              ascii_string line;
                              struct ascii_record *next;
                             } a_record;
typedef struct s_material{
                            struct s_material *last;
                            int atomic_number;
                            int mass_number;
                            float weight_percentage;
                            char library_suffix[5];
                            struct s_material *next;
                           } ll_material;

typedef struct u_list{
                        struct u_list *last;
                        int index;
                        ascii_string label;
                        struct u_list *next;
                      } usage_list;

typedef struct su_list{
                        struct su_list *last;
                        int index;
                        ascii_string label;
                        ascii_string value;
                        char mnemonic[4];
                        ascii_string equivalent_label;
                        struct su_list *next;
                       } surface_usage_list;


ll_material *material_match(a_record *,char[],float * ,int *);
ll_material *memory_s_material(int, ll_material *);
void rollup_llm(ll_material *);
usage_list *memory_usage_list(int,usage_list *);
surface_usage_list *memory_surface_usage_list(int
 ,surface_usage_list *);
usage_list *load_usage_list(char label[],int index,usage_list *);
surface_usage_list *load_surface_usage_list(char[],int
 ,char[],char[],char[],surface_usage_list *);
```

```
void search_usage_list(int,char[],int *,usage_list *);
void search_surface_usage_list(int,char[],int *,char[],char[]
 ,char[],surface_usage_list *);
int mchar(int *,char[]);
void header();
void lines(int);
void bufferpad(char[],int,int);
void abort();
void add_cell(char[],char[],FILE *,int *,int,float,int *);
void add_surface(int,char[],char[],FILE *,FILE *,int *,char[]
 ,surface_usage_list *,int *);
void add_symmetry_surfaces(int,FILE *,FILE *,surface_usage_list *
 ,int);
void add_material(FILE *,int *,char[],int,ll_material *);

void vessel_generation(float apitch,float vod,float vthick,float sod
 ,float sthick,float tutpr,float tcgr,float bltpr,float bcpr
 ,FILE *lu8, FILE *lu9, FILE *lu10,int *ncell,int *nsurface
 ,int *nmaterial,a_record *ptr_core_mtls,char mvessel[],char mshroud[]
 ,char mtg[],char mcp[],char mutp[],char mltp[]
 ,int core_f,surface_usage_list *ptr_surface_usage
 ,usage_list **ptr_material_usage,float bypass_density,float afl){
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -  * v e s s e l _ g e n e r a t i o n *  Generates Vessel Compo-
 - -                                         nents in MCNP Deck
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - Argument(s):
 - -     apitch - fuel assembly pitch                                (input)
 - -        sod - shroud outer radius                                (input)
 - -     sthick - shroud thickness                                   (input)
 - -        vod - pressure vessel outer radius                       (input)
 - -     vthick - pressure vessel thickness                          (input)
 - -      tutpr - Top of Upper Tie Plate Region                      (input)
 - -       tcgr - Top of Core Grid Region                            (input)
 - -      bltpr - Bottom of Lower Tie Plate Region                   (input)
 - -       bcpr - Bottom of Core Plate Region                        (input)
 - - incore_loc - positions for incore instrumentation               (input)
 - -        lu8 - pointer to scratch file for cell representa-       (input)
 - -              tations
 - -        lu9 - pointer to scratch file for surface repre-         (input)
 - -              tations
 - -       lu10 - pointer to scratch file for material repre-        (input)
 - -              sentations
 - -      ncell - number of MCNP gometrical cells in the in-        (output)
 - -              put card images thus far
 - -   nsurface - number of MCNP gometrical surfaces in the         (output)
 - -              input card images thus far
 - -  nmaterial - number of MNCP materials in the problem           (output)
 - -              thus far
 - -  ptr_core_mtls
 - -            - pointer to linked list of core materials          (input)
 - -    mvessel - material identifier for vessel                    (input)
 - -    mshroud - material identifier for core shroud               (input)
 - -        mtg - material identifier for top grid region           (input)
 - -        mcp - material identifier for core plate region         (input)
```

```
 - -       mutp - material identifier for upper tie plate        (input)
 - -              region
 - -       mltp - material identifier for lower tie plate        (input)
 - -              region
 - -     core_f - fraction of core in problem                    (input)
 - -              (1 - full, 2 - half, 4 - quarter)
 - -  ptr_surface_usage
 - -            - pointer to linked list for surface labels     (output)
 - -              and indices
 - -  ptr_material_usage
 - -            - pointer to linked list for material labels    (output)
 - -              and indices
 - -  bypass_density
 - -            - density for bypass water in problem (g/cc)     (input)
 - -        afl - active fuel length (cm)                        (input)
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - -
 - - Variable Definition(s) - - - - - - - - - - - - - - - - - - - - - -
 - - Integer Variable(s)
 - -          ns - counter for number of "virtual" surfaces accmulated
 - -   n_entries - number of elements and isotopes in a particular
 - -               material
 - -       index - index used for searching usage lists
 - -        zero - zero value;
 - -         pid - process identifier of job creating input deck
 - -     version - version number for linkage code
*/
  short int ns;
  int n_entries,index,zero = 0;
  extern short version;
  extern long pid;
/* - Float Variable(s)
 - -  density - material density
*/
  float density;
/* - Character Variable(s)
 - -     buffer - string variable used to load entire lines for print
 - -      label - label for editing
 - -    ptr_buf - pointer to buffer;
 - -       zaid - MCNP element/isotope identifier
 - -   mnemonic - surface type mnemonic
 - - case_title - title for job
 - -      value - definition of surface
 - -     codemn - name of linkage code
 - -      cdate - date of linkage code execution
 - -     crtime - time of linkage code execution
 - - modification_level
 - -            - modification level for linkage code
*/
  char zaid[11], mnemonic[4];
  extern char case_title[];
  ascii_string buffer, label, value;
  char *ptr_buf;
  extern char codenm[5];
  extern char cdate[9];
```

```
  extern char crtime[9];
  extern char modification_level;
/* - FILE Variable(s)
 - - nout - output file
*/
  extern FILE *nout;
/* - Structured Variable(s)
 - -  ptr_mtl - pointer to material definition link list
 - -       pm - working pointer to material definition link list
 - -   ptr_sl - pointer to current record in surface usage link list
 - -   ptr_ml - pointer to current record in material usage link list
*/
  ll_material *ptr_mtl, *pm;
  usage_list *ptr_ml = NULL;
  surface_usage_list *ptr_sl = NULL;
/* - Write Title for Case and Header Records for Cell Representations */
  fprintf(lu8,case_title);
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Input Deck Generated by %s, Version %2i:%c\n"
   ,codenm,version,modification_level);
  fprintf(lu8,"c     Generated on %s at %s by Process %i\n"
   ,cdate,crtime,pid);
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Cell Cards\n");
  fprintf(lu8,"c\n");
  fprintf(lu8,"c     Cells Defining Problem Domain\n");
  fprintf(lu8,"c\n");
/* - Write Header Records for Surface Representations - - - - - - - - */
  fprintf(lu9,"c     Surface Cards\n");
  fprintf(lu9,"c\n");
  fprintf(lu9,"c     Surfaces for Problem Domain\n");
/* - Write Header Records for Material Representations - - - - - - - - */
  fprintf(lu10,"c     Material Cards\n");
/* - Write Header for Deck Generation Edit Table */
  header();
  lines(5);
  fprintf(nout
   ,"0MCNP Input Card Representation Generation Summary Table\n");
  fprintf(nout
   ,"0 Cell Surface(s)     Material/Index/Density    Universe");
  fprintf(nout," Definition\n");
  fprintf(nout,"\n");
/* - Vessel - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cell Cards */
  (*nmaterial)++;
  { int len = 6, length;
    length = mchar(&len,mvessel);
    mvessel[length] = '\0';
  }
  ptr_mtl = material_match(ptr_core_mtls,mvessel,&density
   ,&n_entries);
  *ptr_material_usage = load_usage_list(mvessel,1,*ptr_material_usage);
  ptr_ml = *ptr_material_usage;
  strcpy(label,"Pressure Vessel");
  add_cell(label,mvessel,lu8,ncell,*nmaterial,-density,&zero);
```

```
/* - Surface Cards */
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  strcpy(mnemonic,"pz");
  index = -1;
  sprintf(value,"%10.4E",(tcgr+30.0));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  strcpy(mnemonic,"pz");
  index = 1;
  sprintf(value,"%10.4E",(bcpr-30.0));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Maximum Radial Extent of Problem */
  strcpy(label,"Maximum Radial Extent of Problem");
  strcpy(mnemonic,"cz");
  index = -1;
  sprintf(value,"%10.4E",(vod/2));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Boundaries, if Present */
  if((core_f == 2) || (core_f == 4)){
    (*nsurface)++;
    strcpy(label,"X-Z Plane");
    strcpy(mnemonic,"py");
    strcpy(value,"0.0");
    ptr_sl = ptr_surface_usage;
    while(ptr_sl->next != NULL) ptr_sl = ptr_sl->next;
    ptr_sl = load_surface_usage_list(label,*nsurface,value
     ,mnemonic,"",ptr_sl);
    fprintf(lu8," %i",*nsurface);
    fprintf(lu9,"c     %s\n",ptr_sl->label);
    fprintf(lu9,"   *%1i %s 0.0\n",*nsurface,mnemonic);
    sprintf(buffer,"        %6i",*nsurface);
    bufferpad(buffer,strlen(buffer),59);
    ptr_buf = buffer+59*sizeof(char);
    sprintf(ptr_buf,"%s\n",label);
    lines(1);
    fprintf(nout,buffer);
    if(core_f == 4){
      (*nsurface)++;
      strcpy(label,"Y-Z Plane");
      strcpy(mnemonic,"px");
      strcpy(value,"0.0");
      ptr_sl = load_surface_usage_list(label,*nsurface,value
       ,mnemonic,"",ptr_sl);
      fprintf(lu8," %i",-(*nsurface));
      fprintf(lu9,"c     %s\n",ptr_sl->label);
      fprintf(lu9,"   *%1i %s 0.0\n",*nsurface,mnemonic);}
      sprintf(buffer,"        %6i %s",-(*nsurface),mnemonic);
      bufferpad(buffer,strlen(buffer),59);
      ptr_buf = buffer+59*sizeof(char);
      sprintf(ptr_buf,"%s\n",label);
```

```
      lines(1);
      fprintf(nout,buffer);
  }
/* - Inner Radial Surface of Vessel */
  strcpy(label,"Inner Radial Surface of Vessel");
  strcpy(mnemonic,"cz");
  index = 1;
  sprintf(value,"%10.4E",((vod/2)-vthick));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n= 1.0\n");
/* - Material Cards */
  add_material(lu10,nmaterial,mvessel,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - "Outside" World - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  strcpy(label,"""Outside"" World");
  add_cell(label,"void",lu8,ncell,0,0.0,&zero);
/* - Edit List of Surfaces */
  fprintf(lu8," (");
/* - Maximum Radial Extent of Problem */
  index = 3;
  search_surface_usage_list(0,label,&index,"","","""
   ,ptr_surface_usage);
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Problem */
  index = 1;
  search_surface_usage_list(0,label,&index,"","","""
   ,ptr_surface_usage);
  fprintf(lu8,":");
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  index = 2;
  search_surface_usage_list(0,label,&index,"","","""
   ,ptr_surface_usage);
  fprintf(lu8,":");
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4)){
    strcpy(label,"X-Z Plane");
    search_surface_usage_list(1,label,&index,"","","""
     ,ptr_surface_usage);
    fprintf(lu8,":");
    index = -index;
    add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
     ,nsurface);
    if(core_f == 4){
      strcpy(label,"Y-Z Plane");
      search_surface_usage_list(1,label,&index,"","","""
       ,ptr_surface_usage);
      fprintf(lu8,":");
```

```
      add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
       ,nsurface);
    }
  }
  fprintf(lu8,") ");
  fprintf(lu8,"\n      imp:n= 0.0\n");
/* - Jet Pump Region - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  (*nmaterial)++;
  strcpy(label,"Jet Pump Region");
  add_cell(label,"Bypass Water",lu8,ncell,*nmaterial,-bypass_density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Vessel */
  strcpy(label,"Inner Radial Surface of Vessel");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Outer Radial Surface of Core Shroud */
  strcpy(label,"Outer Radial Surface of Core Shroud");
  strcpy(mnemonic,"cz");
  index = 1;
  sprintf(value,"%10.4E",(sod/2));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
```

```
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/*  - Material(s) */
  { short int i, nloc;
    char *cp;
    strcpy(label,"Bypass Water");
    fprintf(lu10,"c     %s\n",label);
    ptr_ml = load_usage_list(label,*nmaterial,ptr_ml);
    strcpy(zaid,"001001.50c");
    nloc = 0;
    if(*nmaterial < 1000) nloc++;
    if(*nmaterial < 100) nloc++;
    if(*nmaterial < 10) nloc++;
    strcpy(label,"2.0");
    { short int ip;
      for(ip = 1;ip <= nloc;ip++)
        fprintf(lu10," ");
        fprintf(lu10,"m%i %s %s\n"
         ,*nmaterial,zaid,label);
    }
    strcpy(zaid,"008016.50c");
    strcpy(label,"1.0");
    fprintf(lu10,"      %s %s\n"
      ,zaid,label);
    { short int ip;
      for(ip = 1;ip <= (nloc-1);ip++) fprintf(lu10," ");
      fprintf(lu10,"mt%i lwtr.01\n",*nmaterial);
    }
  }
/* - Core Shroud - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  strcpy(label,"Core Shroud");
/* - Load Core Materials Database */
  { int len = 6, length;
    length = mchar(&len,mshroud);
    mshroud[length] = '\0';
  }
  search_usage_list(1,mshroud,&index,*ptr_material_usage);
  if(index == 0){
    (*nmaterial)++;
    index = *nmaterial;
    ptr_mtl = material_match(ptr_core_mtls,mshroud,&density
     ,&n_entries);
    ptr_ml = *ptr_material_usage;
    while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
    index = (ptr_ml->index)+1;
    ptr_ml = load_usage_list(mshroud,index,ptr_ml);}
```

```
    else
      n_entries = 0;
  add_cell(label,mshroud,lu8,ncell,index,-density,&zero);
/* - Surface(s) */
/* - Outer Radial Surface of Core Shroud */
  strcpy(label,"Outer Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  strcpy(mnemonic,"cz");
  sprintf(value,"%10.4E",((sod/2)-sthick));
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
```

```
/*  - Material(s) */
  if(n_entries != 0){
    add_material(lu10,nmaterial,mshroud,n_entries,ptr_mtl);
    rollup_llm(ptr_mtl);
  }
/* - Upper Tie Plate Region - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mutp);
    mutp[length] = '\0';
  }
  strcpy(label,mutp);
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mutp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mutp,index,ptr_ml);
  strcpy(label,"Upper Tie Plate Region");
  add_cell(label,mutp,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","",
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Active Fuel */
  strcpy(label,"Top of Active Fuel");
  strcpy(mnemonic,"pz");
  index = 1;
  sprintf(value,"%10.4E",afl);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Top of Upper Tie Plate Region */
  strcpy(label,"Top of Upper Tie Plate Region");
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",tutpr);
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
```

```
  add_material(lu10,nmaterial,mutp,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Core Grid Region - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mtg);
    mtg[length] = '\0';
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mtg,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mtg,index,ptr_ml);
  strcpy(label,"Core Grid Region");
  add_cell(label,mtg,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Upper Tie Plate Region */
  strcpy(label,"Top of Upper Tie Plate Region");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Core Grid Region */
  strcpy(label,"Top of Core Grid Region");
  strcpy(mnemonic,"pz");
  index = -1;
  sprintf(value,"%10.4E",tcgr);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
```

```
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mtg,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Upper Plenum Region - - - - - - - - - - - - - - - - - - - - - - -   */
/* - Cells */
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,*ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Material not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  strcpy(label,"Upper Plenum Region");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Top of Core Grid Region */
  strcpy(label,"Top of Core Grid Region");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Top of Problem */
  strcpy(label,"Top of Problem");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
```

```
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
/* - Bypass Water Assumed in this Region */
/* - Lower Tie Plate Region - - - - - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mltp);
    mltp[length] = '\0';
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mltp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mltp,index,ptr_ml);
  strcpy(label,"Lower Tie Plate Region");
  add_cell(label,mltp,lu8,ncell,index,-density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Active Fuel */
  strcpy(label,"Bottom of Active Fuel");
  strcpy(mnemonic,"pz");
  sprintf(value,"0.0");
  index = -1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
/* - Bottom of Lower Tie Plate Region */
  strcpy(label,"Bottom of Lower Tie Plate Region");
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",bltpr);
```

```
  index = 1;
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n       imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mltp,n_entries,ptr_mtl);
  rollup_llm(ptr_mtl);
/* - Fuel Support/Core Plate Region - - - - - - - - - - - - - - - - - - */
/* - Cells */
  { int len = 6, length;
    length = mchar(&len,mcp);
    mcp[length] = '\0';
  }
  (*nmaterial)++;
  ptr_mtl = material_match(ptr_core_mtls,mcp,&density,&n_entries);
  ptr_ml = *ptr_material_usage;
  while(ptr_ml->next != NULL) ptr_ml = ptr_ml->next;
  index = (ptr_ml->index)+1;
  ptr_ml = load_usage_list(mcp,index,ptr_ml);
  strcpy(label,"Fuel Support/Core Plate");
  add_cell(label,mcp,lu8,ncell,index,-density,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Lower Tie Plate Region */
  strcpy(label,"Bottom of Lower Tie Plate Region");
  search_surface_usage_list(1,label,&index,"","","""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Fuel Support/Core Grid Region */
  strcpy(label,"Bottom of Fuel Support/Core Plate Region");
```

```
  strcpy(mnemonic,"pz");
  sprintf(value,"%10.4E",bcpr);
  add_surface(0,label,mnemonic,lu8,lu9,&index,value
   ,ptr_surface_usage,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
  add_material(lu10,nmaterial,mcp,n_entries,ptr_mtl);
/* - Lower Plenum Region - - - - - - - - - - - - - - - - - - - - - - -   */
/* - Cells */
  strcpy(label,"Bypass Water");
  search_usage_list(1,label,&index,*ptr_material_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Material not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  sprintf(label,"Lower Plenum Region");
  add_cell(label,"Bypass Water",lu8,ncell,index,-bypass_density
   ,&zero);
/* - Surface(s) */
/* - Inner Radial Surface of Core Shroud */
  strcpy(label,"Inner Radial Surface of Core Shroud");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Symmetry Surfaces, if present) */
  if((core_f == 2) || (core_f == 4))
    add_symmetry_surfaces(core_f,lu8,lu9,ptr_surface_usage,0);
/* - Bottom of Fuel Support/Core Plate Region */
  strcpy(label,"Bottom of Fuel Support/Core Plate Region");
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  index = -index;
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
/* - Bottom of Problem */
  strcpy(label,"Bottom of Problem");
```

```
  search_surface_usage_list(1,label,&index,"","",""
   ,ptr_surface_usage);
  if(index == 0){
    lines(3);
    fprintf(nout,"0***  F a t a l  E r r o r  *** -- Function");
    fprintf(nout," vessel_generation -- \n");
    fprintf(nout," Surface not Found in Linked List, label = ");
    fprintf(nout," %s\n",label);
    abort();}
  add_surface(1,label,"",lu8,lu9,&index,"",ptr_surface_usage
   ,nsurface);
  fprintf(lu8,"\n      imp:n=1.0\n");
/* - Material(s) */
/* - Bypass Water Assumed in this Region */
}
```